Thermal Hydraulic Performance Analysis of a Small Integral Pressurized Water Reactor Core

by

STUART R. BLAIR

B.S. Naval Architecture
United States Naval Academy, 1994

Submitted to the Department of Nuclear Engineering
In Partial Fulfillment of the Requirements for the Degree of

Master of Science in Nuclear Engineering and Nuclear Engineer at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2003

Author .....
Stuart R. Blair, MIT
Department of Nuclear Engineering
July 2003

Certified by .....
Neil E. Todreas, MIT
KEPCO Professor of Nuclear Engineering
Thesis Supervisor

Certified by .....
Mujid S. Kazimi, MIT
TEPCO Professor of Nuclear Engineering
Thesis Reader

Accepted by .....
Jeffery Coderre, MIT
Associate Professor of Nuclear Engineering
Chairman, Department Committee on Graduate Students

Abstract

Thermal Hydraulic Performance Analysis of a Small Integral Pressurized Water Reactor Core

by
Stuart R Blair

Submitted to the Department of Nuclear Engineering on July, 2003,
in Partial Fulfillment of the Requirements for the Degree of Nuclear Engineer

A thermal hydraulic analysis of the International Reactor Innovative and Secure (IRIS) core has been performed. Thermal margins for steady state and a selection of Loss Of Flow Accidents have been assessed using three methodologies to account for uncertainty. The thermal hydraulic analysis has shown that the IRIS is designed with adequate thermal margin for steady state operation, the locked rotor/shaft shear accident (LR/SS) and for variants of the partial loss of flow accident. To treat uncertainties, three methods were used, ranging from conservative, deterministic methods, to more realistic and computationally demanding Monte Carlo-based methods.

To facilitate the computational requirements of the thermal hydraulic analysis, a script-based interface was created for VIPRE. This scripted interface (written in Matlab) supplants the existing file-based interface. This interface allows for repeated, automatic execution of the VIPRE code on a script-modifiable input data, and parses and stores output data to disk. This endows the analyst with much greater power to use VIPRE in parametric studies, or using the Monte Carlo-based uncertainty analysis methodology. The Matlab environment also provides powerful visualization capability that greatly eases the task of data analysis.

Thesis Supervisor: Neil E. Todreas
Title: Korea Electric Power Company Professor of Nuclear Engineering, Professor of Mechanical Engineering

Acknowledgements

I would like to thank Professor Neil Todreas for his patience and encouragement through the course of this work.

I would like to thank the engineers at Westinghouse Electric Co. Science and Technology Division, in particular Luca Oriani, for their substantial contributions without which this research would not have taken place.

I am most endebted to my wife Catherine and kids Joshua, Jordan, and Jolene for their patience, understanding and support.

In Loving Memory of My Father

Jerold Lee Blair
(1944-1995)

Stuart R. Blair

July 21, 2003

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1. Objectives

This thesis has two complementary objectives. The first is to present a thermal-hydraulic (T/H) analysis of a small, integral, pressurized water reactor (PWR) core. The second is to introduce a set of computational tools that are designed to both expedite the T/H analysis process as well as harness the power of modern computer hardware to allow more detailed and comprehensive studies by providing a script-based interface to the T/H analysis program VIPRE.

1.2. The IRIS Reactor

General Description

The IRIS is a modular, integral, light water cooled, low-to-medium power (~ 350 MWe) reactor, which emphasizes proliferation resistance and enhanced safety to meet the requirements defined by the U.S. Department of Energy for Generation IV reactors. A distinguishing characteristic of the IRIS is the integral design: The steam generators (S/Gs), reactor coolant pumps (RCPs) and pressurizer (PZR) are all contained within the reactor pressure vessel (RPV). This configuration is quite obviously very different from a conventional PWR where the S/Gs, PZR, and RCPs are all mounted outside of the RPV, connected by reactor coolant piping of varying diameter, all located within a containment. A simplified diagram of the IRIS RPV is shown in Figure 1.

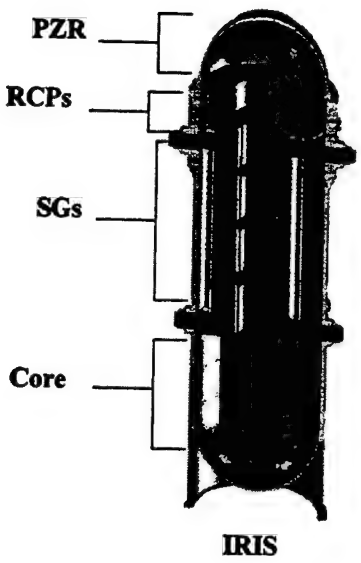


Figure 1: IRIS Integral Reactor Configuration.

Over the past decades there have been several projects involving the integral reactor concept. Many of the projects, as with many new reactor design projects in general, had terminated early in the conceptual, paper stage, while others have endured as viable entities for many years. Several integral reactor designs have been presented in the open literature.[a] A summary of past and current integral reactor projects is presented in Appendix 1. Advantages extolled by the designers of integral reactors include increased safety, more compact layout and reduced construction costs.

Increased safety for integral reactors comes from one or a combination of the following design features: low power density, passive safety features of the containment, natural coolant circulation, and the central feature of the integral core configuration - no reactor coolant loops. The elimination of all reactor coolant piping removes that piping from loss of coolant accident (LOCA) possibility, and the probabilistic risk assessment (PRA) can ignore such breaks as initiating events. In the case of the CAREM-25, a preliminary PRA was carried out that indicated that due to the aforementioned design features the core damage frequency is two orders of magnitude less than that of a typical PWR, with attendant consequences three orders of magnitude less.[1]

[a] In this context, 'open literature' means: English language, non-proprietary publications.

The compact plant layout is derived primarily from the elimination of the reactor coolant piping and by placing equipment normally external to the RPV such as the S/G and PZR within the vessel. The elimination of the requirements for large on-site welds on reactor coolant piping, as well as the modular configuration of the reactor vessel assembly, is expected to lead to a shorter construction time. This, in conjunction with the overall smaller physical footprint, is expected to lead to lower construction costs.

Thermal Hydraulic Analysis of IRIS

A key step the design of a new reactor plant is the core T/H analysis. In this step, it is determined whether or not the fuel can satisfactorily be cooled under both nominal and accident conditions.

From a T/H perspective, the IRIS core is somewhat of a departure from typical PWRs in that core power, specific power, linear power and power density are all lower than that of typical PWR designs. Some T/H parameters are provided for comparison in Table 1.

	Standard PWR	IRIS	AP1000	AP600
P/D	1.32	1.4	1.33	1.33
Core power (MWth)	3411	1000	3415	1940
Specific power (kW/kgHM)	40	20.72	40	29
Linear power (kW/m)	18	12.9	18.8	13.2
Power Density (kW/l)	106	51	110	79

Table 1: Comparison of Thermal Hydraulic Parameters of IRIS to Typical PWR Designs.

The core T/H analysis includes a study of steady state behavior, a sensitivity study that examines the IRIS model response to perturbations of key operational parameters, transient analysis of three types of LOCA events and a hot channel analysis that incorporates the uncertainty inherent in plant design parameters and combines them to provide conservative predictions on plant behavior during a set of transients.

The hot channel analysis plays a central role to the core T/H analysis in that it is the means by which design uncertainties are translated directly to evaluate the impact of such

uncertainty on the T/H acceptability of the design. Three separate methodologies are employed to perform this hot channel analysis in which different rules are applied in the combination of the uncertain parameters, as well as in the interpretation of the results.

1.3. Computational Tools

The number of computations that are required in a typical nuclear reactor core design is enormous. The invention of digital computers, their employment in nuclear design applications, and gradual integration into mainstream engineering use, has lightened the manual 'pen-and-paper' calculation required by the ordinary design engineer considerably. Software tools[b] constitute the interface between the engineers and the ever burgeoning computer power that is capable of performing billions of calculations every second. Software designed specifically for the purpose of T/H analysis allow ordinary engineers to harness this computational power and perform their job – designing the nuclear reactor core - much more quickly than what was possible in the past.

Despite these advances, codes designed for performing T/H analysis, and consequently the engineers who use them, often fail to fully exploit the capabilities of the machine on which they are being run. In many cases, this is a result of the way in which the engineer is forced to provide inputs to the software, as well as the way in which the program communicates its results with the engineer. Text files with strict and torturous formatting rules are required to communicate program inputs. Output data is buried within a dizzying barrage of numerical results. The cheap, ubiquitous computers – each endowed with tremendous power – are largely idle while engineers, whose speed of work has *not* improved by large orders of magnitude during the computer revolution, chiefly spend their time trying desperately to create an input file that *actually runs* and labor furiously to create a single attractive plot from the mountain of output data.

In this thesis, a set of software tools: a script-based interface - is introduced and demonstrated. Using these scripts, input files, analysis code execution, output data parsing, and other related tasks are performed 'automatically' in accordance with 'scripted' logic. These scripts, to a large extent, allow the analyst to work more

[b] In this thesis, 'Software tools' will often be colloquially referred to as 'codes'.

productively with existing analysis codes to produce useful results more quickly. Additionally, because of the nature of the script-based interface, analyses can be conducted in ways that would have been completely infeasible without such an interface.

1.4. Thesis Organization

Following this introduction, Chapter 2 gives a thorough introduction to the Matlab-VIPRE Interface. This script-based interface is shown to be a more powerful and versatile tool than that which is customarily provided with scientific software. Included is a detailed application of this interface to the IRIS Tight core design. Additionally ideas are developed for extension of the Matlab-VIPRE Interface to include other programs such as the fuel performance analysis code FRAPCON. A more complete documentation of the implementation details of the Matlab-VIPRE Interface is given in Appendix 2.

Chapter 3 is a report on the thermal-hydraulic analysis of the IRIS Open core design. In addition to the detailed steady-state analysis, a complement of LOFA casualties are analyzed including a complete loss of flow accident (CLOFA), a partial loss of flow accident (PLOFA) where four of the eight installed reactor coolant pumps cease operating, as well as the locked rotor/shaft-shear (LR/SS) accident where only one reactor coolant pump is incapacitated. The transient analysis is enhanced by a thorough uncertainty analysis where two established methodologies: the standard thermal design procedure (STDP) and the improved thermal design procedure (ITDP) are employed along with a third, relatively unestablished procedure: the Monte Carlo uncertainty procedure (MCUP) is used to provide a more thorough and accurate, albeit computationally demanding accounting of the inherent uncertainties of the core design.

Chapter 4 presents the conclusions from the thermal-hydraulic analysis. In this chapter, a qualified determination is made outlining the extent to which the IRIS Open core satisfies thermal limits. A comparison is made between the conclusions that would be drawn from the use of different uncertainty analysis procedures which also outlines advantages and disadvantages of each.

Chapter 5 provides an outline of recommended future work. Shortcomings in the present thermal hydraulic analysis procedure are highlighted along with potential avenues

for rectification. A discussion is given to possible future directions of the Matlab-VIPRE Interface along with related computational components. A bibliography is provided in Chapter 6.

Following the main body of the thesis is an extensive set of appendices that expound upon certain technical topics that are related to discussions present in the thesis proper. Prospective users and developers of the Matlab-VIPRE Interface are provided with a more detailed description of the interface implementation details in Appendix 2. Extensions to codes such as FRAPCON and RELAP, and practical direction regarding script usage and implementation is provided in Appendix 2 – Appendix 8. Appendix 9 is given as a preliminary fuel cycle cost analysis of the IRIS Open core. This analysis is intended to be the starting point of a more thorough and comprehensive economic analysis of theIRIS fuel cycle.

Chapter 2 Thermal Hydraulic Analysis Tools

2.1. Introduction

For the purposes of performing a full thermal hydraulic analysis of the IRIS Open[a] core, it is necessary that a collection of computational tools be utilized. While paper-and-pencil computations are a necessary and important part in understanding local channel phenomena, the complexity that accompanies detailed, predictive calculations does not allow hand calculations. For this project, the VIPRE 1-mod 2 sub channel analysis code (reference [2]) was used to evaluate in-core channel conditions during steady state and transient conditions. For transient calculations, the system simulation code RELAP[3] was utilized to provide necessary core boundary conditions.

VIPRE and RELAP are codes typically used in the thermal hydraulic design of a nuclear reactor. They are both programmed almost entirely in standard Fortran 77, and are interfaced via ASCII text-based files for both inputs and outputs. The text-file based interface is common for many other codes used in reactor design such as: CASMO[4], SIMULATE, MCNP[5], and FRAPCON[6]. This list could easily be expanded to include dozens of other codes developed by nuclear engineers, as well as hundreds of others

[a] Here, the word OPEN is used to distinguish this core design from an alternative: IRIS Tight, that will be introduced in this chapter.

crafted among the scientific community in general. These codes are an embodiment of an enormous quantity of experimental data and theoretical knowledge, custom tailored to the solution of problems in nuclear reactor design.

This chapter will focus on the inadequacies inherent with the aforementioned file-based interface to analysis codes, and the script-based interface that was created as a replacement.

2.2. User Interface

For any given software package, in many ways, the user interface *is* the code. As technical software users, we customarily spend a good deal of time learning the general principles involved with the internal workings of the code. Despite this, there are plenty of engineers who use complex codes such as CASMO or SIMULATE without being able to write down anything that represents in a significant way the sets of equations being solved internally to the code. The thing that matters most to the user is: What can this code tell me? Practical considerations also include: How do I ask the code questions? And: In what ways are the answers provided to me? The above questions can be considered to be a good definition of the code user interface. In this section, I discuss various means that codes have, over time, been designed to interact with users - the interface, and the impact that this interface has on usability of the code.

Command-Line Arguments

Command line arguments represent, probably, the most basic way that a user can customize the behavior of a compiled program without making changes to the source-code itself. Programs can be extended in this way to allow more usefulness and flexibility. A typical example of this interface familiar to many engineers would be:

g++ –o my_program my_program.cc –Imy_includes/

This is a command to invoke the GNU-C++ compiler from a LINUX Bash shell. The command line arguments tell the compiler to create an executable called

my_program from the C++ source file *my_program.cc* adding to the search path for header files the directory *my_includes*.

In principal there are really no limitations on how much information can be exchanged between the user and the code via this interface. The practical constraints, however, are many. The user can only reasonably be expected to manually enter a small number of command-line arguments, or 'options', with each execution of the code. Secondly, these arguments do not persist between code executions. They must be re-entered each time.[b] While there are likely further operating-system-dependent limitations, it is clear that there are severe practical constraints on the user interface that is provided only on the command line. Similar limitations exist if the program output is to be directed only to the user terminal. These constraints lead programs to migrate towards a more flexible, file-based interface.

File-Based Interface

The file-based interface has its roots in the very earliest days of stored-program computers. Text files, almost always with a very rigid format, were required to provide input data for the code to process. In a similar style, the fruits of the program's labors would be deposited in a file for the user to sift through and glean desired information.

This program has the advantage that a great deal of input data can be provided allowing for extensive customization of program behavior. Both the input and output data are persistent, in that it does not disappear between program runs.

This approach has several disadvantages. First, the format required for the input data is typically such that a considerable amount of time and care must be put into the preparation of the input data. This is effort beyond that required for the formulation of an appropriate physical model. In addition to choosing appropriate assumptions, geometric discretization, and physical correlations, the user is required to encode these selections to the precise specification of the application program. A user making any syntactic

[b] Though, perhaps it is worth noting that the example given is an example where a more useful interface is provided: a Makefile – a special file that is read by a program called *make* which automates the building process (i.e. preprocessing, compiling, assembling, linking) of creating an executable program from a C++ source file. This system is, for all intents and purposes, a script-based interface with the Makefile as the (often automatically generated) script.

mistakes, even those as simple as a missing comma or capitalized letter, can expect abrupt program termination accompanied with characteristically cryptic error messages. Finding and correcting mistakes in the input file can take anywhere from hours to weeks. Secondly, and as a corollary to the first, most modifications to the input data would require a similarly large amount of tedious work – devoid of scientific reward or gratifying insight – with little benefit gained from having done this work already. As an example, for the VIPRE input file, a conceptually trivial change to core geometry such as an adjustment in the fuel pitch-to-diameter ratio, results in hours spent revising the input file. Thirdly, these modifications have to be performed manually. No hope of automation is provided.

The file-based output format is not without disadvantages of its own. VIPRE produces, for the benefit of the user, a very detailed collection of information regarding the calculated state of the working fluid in each sub-channel. For the typical simulation that was performed for the IRIS Open core, the output file for a steady-state calculation would reach approximately 800 KB in size. While this is not terribly large, it is more than is easily dealt with in the case of printed output. In addition, as text, it is inconvenient to transfer this data to an effective visualization or data analysis program.[c] For more complicated situations such as a transient, the output data file could grow to be as large as 200 MB. At this time, it is beyond the capabilities of ordinary text-editing programs merely to open the file to view its contents, to say nothing of the analysts' ability to actually manipulate this data and form useful conclusions.

Graphical User Interface

At the present time, most respectable, newly authored, commercial and industrial purpose codes come equipped with a graphical user interface (GUI) of familiar design. Though the details of this interface vary among different operating systems, most packages conform to a standard of menu layout and task organization that have made user experiences intuitive and easy to learn.

[c] Despite any inconveniences, scientists have predictably become very adept at writing simple, custom programs designed to extract desired data from text-based output files.

Modern GUIs solve many of the problems associated with the file-based interface. Input data can often be entered in an interactive, visual environment. Choices for input variables can be limited, with only valid choices available. These environments focus heavily on ease-of-use and reduce considerably the time lost in the model formulation stage of the design. Often, provisions are made to allow for especially simplified execution of routine tasks. Output data is commonly post-processed and presented to the analyst in graphs, plots and other useful ways so that incredible insight can be gleaned from enormous data sets that a decade ago would have been completely unmanageable.[d]

Scripted Interface

The alternative to a file-based or GUI presented in this chapter is the script-based interface. A script-based interface is designed in such a way, that the inputs and outputs of a particular code, like VIPRE, are provided in the form of programmed function inputs and outputs. The VIPRE code itself becomes a functional part of a separate code (script) written by the analyst with the aim of performing the desired analysis. Examples of a script-based interface development for other codes exists in the recent literature.[7],[8]

A real shortcoming in both GUIs and file-based-interfaces is that they restrict the *kind* of analysis that can be done. Put differently: they restrict the *types* of questions that an analyst can ask. It is interesting that such a restriction would have its roots in the user interface, rather than in the code itself – but this seems completely to be the case. VIPRE is capable of answering many interesting questions: What is the maximum temperature at the centerline of the hottest rod? How close is the core to DNB? What is the pressure drop across the core? What is the fluid flow velocity in the core? These questions can be answered and communicated using the file-based interface as well. But the analyst, with a file-based interface, cannot, in any reasonable amount of time determine:

> What is the maximum allowable core power such that I do not exceed a 60-psi pressure drop, keep core outlet quality below 5 percent, and restrict flow velocity to below 10 m/s, while maintaining MDNBR above 1.4? How will this change as I adjust the fuel lattice pitch and rod diameter?

[d] Scientific data visualization has become a sub-field of scientific computing in and of itself.

But, these are *exactly* the sorts of questions that an engineer needs to answer for a thorough thermal hydraulic design – and VIPRE is fully capable of providing the answers provided that it is asked in the right way. The script-based interface was created for this purpose.

In this instance, the script-based interface is built on top of the file-based interface. The file-interface is maintained, but the input file is automatically generated by the script-interface based on information programmed by the analyst. Similarly, through minor modifications to the VIPRE source code, key computational outputs are directed to data files that are easily parsed by the script interface and presented to the analyst (programmer) via data structures in the scripted language.

2.3. Matlab-VIPRE Interface

General Description

The particular script-base interface created for this project will be referred to simply as the Matlab-VIPRE Interface. The scripting language chosen is Matlab.[e] As an aid to the programmer in assembling and manipulating input data, the ubiquitous spreadsheet program Microsoft Excel[f] is also used. A schematic representation of the Matlab-VIPRE Interface is provided in Figure 2.

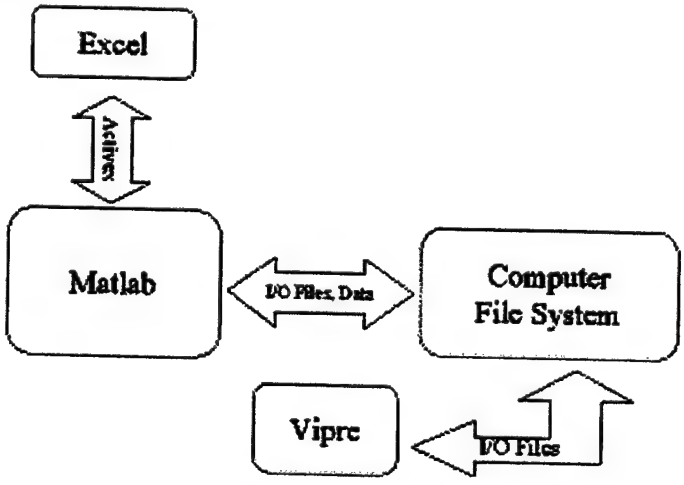


Figure 2: Schematic representation of the Matlab-Vipre Interface

[e] Matlab is a registered trademark of The Math Works, Inc. Natic, MA.
[f] Excel is a registered trademark of Microsoft Corporation, Redmond, WA.

Before the Matlab interface was created, a usual way that the analyst organized the input data was by using Excel. Typically, all of the input data would be carefully structured in the spreadsheet in such a way that much of the actual input file for VIPRE could be literally cut-and-pasted out of the spreadsheet. These 'Excel-based interfaces' are often fairly sophisticated in that many of the calculations required to implement changes in rod diameter or lattice pitch are propagated through spreadsheet formulae. This greatly expedites any core geometry changes that must be made. A screen-shot of this typical interface is provided in Figure 3

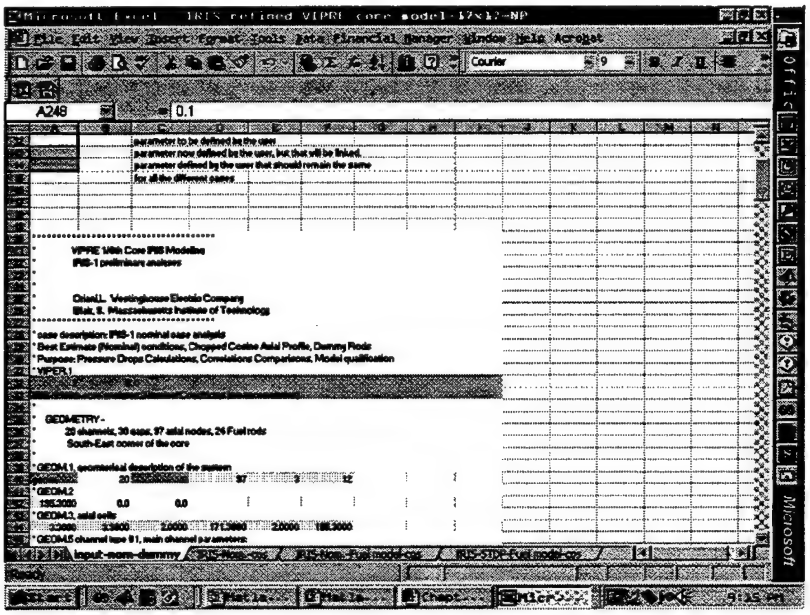


Figure 3:Microsoft Excel Spreadsheet With VIPRE Input Data.

Implicit in the initiation of a large effort to provide a script-based interface to VIPRE is the desire to minimize the amount of work that must be done. At the outset, no coherent design existed, primarily due to the fact that there were many unanswered questions as to how everything would be done. First and foremost was the question of: What language will be used for the scripting interface? How will this interface be tied into VIPRE? How will the outputs of VIPRE be dealt with? Each of these questions was dealt with individually, but the answers all rested on some key early decisions. The first of which was that the initial input for the scripted interface, would be provided through Excel.

For the IRIS open design, scientists at Westinghouse had, as of the spring of 2002 spent a considerable amount of time in developing an acceptable VIPRE model of the IRIS core. This model was organized in a set of Excel spreadsheets which together, greatly simplified the process of generating a correct input file for VIPRE. The detailed geometric description of the rods, channels, correlation selections, power profile specifications, and coefficient calculations were performed on separate worksheets. The data provided was then collected on a single worksheet that was structured in the required format for a VIPRE input file. Conversion of this worksheet to a valid input file consisted of systematically converting this file to an ASCII text file.

The initial aim of the work was to devise a way to somehow automate the process of converting this worksheet to a text file. Matlab was chosen for this task. The primary reason for this choice was the powerful assortment of both high and low-level Input/Output (I/O) facilities provided by this program as well as extensive interoperability with other Windows[g] programs[h] via ActiveX technology. So the decision was made that Matlab should somehow be used to automatically extract the input data from Excel, and then generate a text file with exactly the correct format as required for VIPRE input.

Excel	Matlab
Several Worksheets Containing Model Parameters and Data	Communicate with Excel via Activex Controls to extract input data
Perform simple computations to adjust the input data based on model parameters: e.g. re-compute channel flow area after changing pitch or diameter	Using communication pathways, modify model parameters such as channel pitch or diameter as required by script.

Table 2: Matlab and Excel Interaction in Matlab Vipre Interface

Conveniently, Matlab has, through the use of ActiveX controls, the built-in ability to address a specific Excel spreadsheet. The resulting object within the Matlab workspace

[g] Windows is a registered trademark of Microsoft Corporation.
[h] For example: Excel

context provides handles to all of the spreadsheet data and functionality. Low-level functions are thus available for extracting data from the spreadsheet, directly into the Matlab workspace. A typical set of function calls using this capability are:

```
[xl,xl_wb_specific] = get_spreadsheet(filename);
input_sheet = get(xl_wb_specific,'Sheets','input');
vipre1.kase = get_cell_value(input_sheet,'A32');
geom3(i).dx = get_sheet_row_col(input_sheet,44,index);
```

Once this data is extracted, it is saved to Matlab-style data structures very analogous to the C-type *struct*. A detailed script was written to carry out this action, therefore, the action of extracting all of the required input data from a specified Excel workbook, is reduced to the invocation of a single script.

Next it was required to convert the data, now in the Matlab environment, to a text file ready to be used by VIPRE. Communication between the user-written script and VIPRE would take place entirely through the file system.[i] Another script was written specifically for this purpose. Low-level file I/O facilities are provided with Matlab very similar to equivalent functions in C. Each input field to be provided to VIPRE, referred to as a 'card', is written to the text file, with the correct format, and in the correct order. Separate low-level functions were written that would write the data for each individual field. A larger script was written to orchestrate the entire process.[j] In this way, having extracted the input data from Excel, this function would carry out all actions required to generate the required input file and write the file to the hard-disk. Additionally, using the

[i] i.e. data to be transferred from the Matlab workspace to the VIPRE input context and vice versa would first be written to a file by one program, and then read from the file by the other.

[j] This larger, orchestrating script is required to enforce the required order of input in VIPRE. The individual input cards in VIPRE are not identified. Rather, only a single card at the beginning of each 'section' of the VIPRE input. (Example: the first card of the geometry section has, as the first input variable, the flag 'geom' – in this way, the start of the geometry section is identified.) Once the input section is identified, the remaining cards within the individual section follow in an order that can be dynamically determined from the cards as they are read within that section. The data must be entered in a way that preserves this order, or an input error will be produced by VIPRE.

high-level I/O of Matlab, the input data itself could easily be stored to hard-disk and recovered,[k] allowing the same input data to be used to generate multiple input files.

It is straightforward to manipulate the input data within the Matlab environment simply by variable initialization and assignment. In this way, the input data itself could be changed within the Matlab environment prior to writing it to disk. As an example of this important capability, The following command changes the core mass flux to $2.5\,Mlbm/ft^2-hr$:

```
change_input(var_file_name,'oper5.gin',2.5)[l];
```

This capability is fundamental for many of the capabilities that a script-based interface should provide. More significantly, these manipulations to the input data, and consequently the input file, could be scripted and automated, so that the input data could be adjusted automatically any number of times by high-level scripts within Matlab. This created the groundwork for multiple, systematic VIPRE simulations.

These systematic VIPRE runs are the key to the power of the script-based interface. VIPRE takes on many of the characteristics of a built-in Matlab function, so more general algorithms developed and implemented using the Matlab language can easily be adapted for use with VIPRE. If one wants to perform a general parametric analysis, all of the capability is present to automate the systematic coverage of the parameter space desired. If one wants to perform an optimization study of a particular core, the ability to manipulate the input data for VIPRE and examine the resulting output is exactly the behavior required.

Once the input file is created, it is then required that the input file be provided to VIPRE for processing. The method for this is simply to place the input file in the same

[k] Using the 'save' and 'load' functions provided in Matlab, this process could be accomplished in a single line of code.

[l] The first argument is the name of the '*.mat' file that holds the input data, the second argument must be a string variable that is the specific variable to be modified, the third argument is the value that the variable is to take. Using this function preserves the input data file in the order required for proper input file generation.

directory as VIPRE.[m] The input file is named, logically, "input.txt", and the executable is invoked. Assuming that the program runs properly and no errors arise, a set of output files are created by VIPRE and deposited in this same directory. The sequence of events is:

- Matlab extracts input data from Excel spreadsheet and writes a valid input file to hard-disk.
- Second, through built-in functions, Matlab copies this input file to the directory where the VIPRE executable resides.
- Matlab causes VIPRE to be invoked, and then dispatches in an appropriate fashion, the output data generated.

Matlab version 5.3 has no built-in function that invokes an outside executable directly.[n] Matlab has been, however, endowed with the extensibility feature whereby C/C++ or Fortran functions can be written, compiled as dynamic link libraries, and the resulting functions called directly from within the Matlab environment. The solution is thus, to write a trivial C program that makes the proper operating system call to invoke the VIPRE executable, and then call this function from within Matlab. All that is left then, is the processing of the output data that is also performed by a script.

Several minor modifications of the VIPRE source code were required for the completion of this research. They are simply enumerated here[o]:

- Several new output files were created that would contain only the numeric data generated from various routines within the VIPRE program. This was to simplify the task of parsing the output data from within Matlab
- The input and output files were changed so that they had the '.txt' postfix to allow simplified handling within the Windows environment.
- The maximum allowed number of transient time-steps was increased to 500 from the previous limit of 10.

As can be seen, these modifications are all fairly straightforward in nature and had no significant impact on the running of the code and, in particular, in the reliability of the results produced. The last modification was absolutely essential as the transients we hoped to run were discretized into over 400 time-steps.

---

[m] This is exactly what one would using only the file-based interface.

[n] Version 6.5 now has the capability of directly calling the operating system by re-directing command line-inputs to the operating system shell.

[o] Detailed descriptions and documentation of these changes is provided in Appendix 2.

Low-Level Routines

Numerous low-level routines were required for the proper operation of the interface. Each input-card for the VIPRE input deck is written to the input file individually. The routines that complete this task are typical of what one would expect for a C or Fortran statement for writing to a file. For example, the following is a function for writing the VIPRE input card: OPER.5 to the input file:

```
function gen_oper5(fid,oper5)

pref = sprintf('%7.4f,',oper5.pref);
hin = sprintf('%7.4f,',oper5.hin);
gin = sprintf('%7.4f,',oper5.gin);
pwrinp = sprintf('%7.4f,',oper5.pwrinp);
if oper5.hout == 'empty'
  hout = '0';
else
  hout = num2str(oper5.hout);
end
entry = strcat(pref,hin,gin,pwrinp,hout);
count = fprintf(fid,'%s \n',entry);
```

The function is provided a handle[p] to the VIPRE input file: 'fid', and a copy of the Matlab variable holding the input data for the VIPRE input card 'OPER.5'. The function merely formats and writes this data to the hard-disk. Similar routines are provided for other VIPRE input cards.

Low-level routines were also required for communication between Matlab and Excel. In particular, it became necessary to have easy ways to obtain an interface to an existing Excel worksheet,[q] 'grab' data from cells within that worksheet, and modify cell entries from within the Matlab environment. These features are critically important to the integrity of the scripted interface. If they were not provided, calculations would have to be interrupted while changes are manually made to the spreadsheets. In general, any operation that cannot be performed via the Matlab script, fundamentally limits the types of analysis that one can do – the limit being the requirement for manual intervention in the analysis process. Tasks that are customarily carried out by hand – steam-table look-

[p] In this context word 'handle' means a type of variable through which a referenced object (data structure, program, function) can be accessed and manipulated.
[q] In this thesis, the word 'worksheet' will be used to refer to a particular sheet within an Excel workbook. The word 'workbook' is used to refer to the Excel file (e.g. my_work.xls).

ups, graph-reading and changes to spreadsheets – *must* be automated, or the interface will be incomplete and the expected benefits will not be obtained.

High-Level Routines

Following the development of the low-level Matlab scripts to carry out the necessary tasks to perform an individual VIPRE run, higher-level scripts were created to perform common tasks, some of which have already been mentioned. These scripts are formed by cobbling together the various lower-level tools and are actually quite powerful. Some of these scripts are enumerated here[r]:

- Get_excel_input.m: this is actually a fairly low-level script, but is powerful in that, provided the arguments of a specific Excel workbook (and worksheets) where the input data resides, all of the input data is extracted from the specified workbook, and stored to disk in Matlab style data structures.
- Generate_input_file.m: This script calls the necessary low-level functions required to generate a complete VIPRE input file.
- Parse_viper_output.m: This script post-processes the VIPRE generated output files and places the data in Matlab style data structures and stores the data to disk.
- VIPRE_run.m: This script carries out the task of copying the input file to the directory where the VIPRE executable resides, calling the required function to invoke VIPRE, and calls Parse_viper_output.m.

It should be clear to the reader, that the functionality provided by these scripts allow one to quickly pre-program ('script') a sequence of useful VIPRE activities. In all cases, input files are created without the usual human flaws,[s] and there is no human intervention required to collect the output data. In addition, Matlab is endowed with powerful plotting functions that make visualization of the results relatively simple. This eliminates the tedious work of extracting the data from the voluminous output files, and allows more time for actual analysis. Several Matlab-VIPRE Interface applications are provided in Appendix 2, along with the entire source code for the Matlab-VIPRE Interface.

[r] Full listings of the code for these functions are provided in Appendix 2.
[s] Missing commas, misplaced decimal places, etc…

2.4. Modelling the IRIS Tight Core With The Matlab-VIPRE Interface

The Matlab-VIPRE Interface described in this chapter provides a potent tool for dealing with tedious and error-prone tasks associated with computational analysis: input file preparation and modification, gathering data from output files and visualizing results. As an example of the power and convenience of these features, a short analysis is conducted to explore some of the behavior of the IRIS Tight Core design.[9],[10]

The IRIS Tight Core design is intended to be an upgrade to the current IRIS design, referred to as IRIS Open. The IRIS Tight Core design is so called because of the relatively 'tight' geometric configuration of the fuel assemblies, with a currently designed pitch-to-diameter ratio (P/D) of 1.156. This design is intended to generate more power, have a longer cycle length and comparatively favorable economics compared to the reference IRIS Open design, while maintaining the same physical core size. Relevent design characteristics of the IRIS Tight core as compared to the IRIS Open and a Standard PWR are given in Table 3.

	Standard PWR	IRIS OPEN	IRIS TIGHT
P/D	1.32	1.4	1.156
H/HM	3.4	4	1.29
ODROD (mm)	9.5	10.74	9
Enrichment (w/o 235U)	3%-5%	~ 5%	~14%
Bu (MWD/KgHM)	~50 (Reactivity Limited)	40 (Reactivity Limited)	70 (Metallurgically Limited)
Core power (MWth)	3411	1000	1661
Specific power (kW/kgHM)	40	20.72	25
Linear power (kW/m)	18	12.9	11
Power Density (kW/l)	106	51	65

Table 3: Comparison of Core Characteristics. (Source: [10])

Notice that, in this table, there is no specification of the fuel assembly lattice pitch other than through the given P/D and rod outside diameter. Thermal hydraulic analysis

presented in reference [10] demonstrates that for the nominal rod diameter of 9mm and the operational conditions given in Table 4, the MDNBR is above a prescribed thermal limit of 1.50.[t]

Parameter	Value
Linear Power	**11 kW/m**
Coolant Mass Flow Rate	**18.01 lbm/sec (70 % nominal for 1/6th assembly)**
System Pressure	**2220 psia**
Core Inlet Temperature	**561.6 °F**

Table 4: Operational Parameters for IRIS Tight Thermal Limit Calculation.

The question to be answered is: how does the computed MDNBR change as fuel rod outside diameter is changed. This analysis incorporated the following additional constraints:

- The P/D is maintained constant at 1.156 by adjusting the rod pitch to compensate for each rod diameter change.
- The guide-thimble diameter is adjusted to be the same as the fuel rod diameter in all cases. This ensures that a reasonable configuration will be maintained in channels with juxtaposed fuel rods and guide-thimbles.[u]
- The wire-wrap diameter is adjusted so that the wire maintains contact with adjacent fuel rods. After the rod diameter and pitch are adjusted, the wire wrap diameter is set to be equal to:

[t] The detailed development of the rationale for the given thermal limit is given in reference [10].
[u] I.e. the fuel rods and guide thimbles will not geometrically overlap – as could happen in the case where the fuel rod diameter is reduced, then pitch is reduced so that the fuel rods and guide thimbles are brought closer together. Since the P/D > 1, the pitch will be reduced by a larger amount than the fuel rod diameter is reduced, therefore rod overlap is possible if the guide thimble diameter is not also reduced. The guide-thimble is, for the reference design identical in diameter to the fuel rod. For simplicity, this relationship is maintained.

Equation 1

$$D_{wire-wrap} = P - D_{fuel\_rod}$$

where:

P	= pin pitch
D_{fuel_rod}	= fuel rod diameter

- The specific power is maintained constant by adjusting the linear power with each rod diameter change in accordance with the following:

Equation 2

$$q' = \alpha \rho_{HM} \frac{\pi D^2_{fuel\_pellet}}{4}$$

Where:

q'	is the Linear Power
α	is the Specific Power
ρ_{HM}	is the fuel density
D_{fuel_pellet}	is the fuel pellet diameter

The specific power is to be held constant, and the fuel density is assumed constant, therefore the linear power can be equivalently expressed:

Equation 3

$$q' = cD^2_{fuel\_pellet}$$

where: $c = \frac{\alpha \rho_{HM} \pi}{4}$

To adjust the linear power to maintain constant specific power with different rod diameters therefore:

Equation 4

$$\frac{q'_o}{D^2_{fuel\_pellet\_o}} = \frac{q'_{new}}{D^2_{fuel\_pellet\_new}} = c \Rightarrow q'_{new} = q'_o \frac{D^2_{fuel\_pellet\_new}}{D^2_{fuel\_pellet\_o}}$$

- The coolant mass flow rate is adjusted to maintain a constant enthalpy rise across the core in accordance with the following:

Equation 5

$$q'L = \dot{m}\Delta H$$

where:

L	is the channel length (ft)
$\dot{m}$	is the channel coolant mass flow rate (lbm/sec)
ΔH	is the channel coolant enthalpy rise (BTU/lbm-°F)

For channels of constant length, and constant enthalpy drop: $\Delta H/L$ is a constant. An expression for mass flow rate as a function of the new and reference linear power is therefore:

Equation 6

$$\frac{q'_o}{\dot{m}_o} = \frac{\Delta H}{L} = c = \frac{q'_{new}}{\dot{m}_{new}} \Rightarrow \dot{m}_{new} = \dot{m}_o \frac{q'_{new}}{q'_o}$$

According to the above procedure, the fuel rod diameter is perturbed from the nominal value of 9mm over the range: 8.5mm to 9.7mm.

To perform this analysis by hand would an arduous task. Though it is straight-forward to perform the computations outlined in

Equation 1 to Equation 6, the modifications to the core geometry result in extensive changes to the VIPRE input file. Every channel characteristic such as flow area, heated and wetted perimeter must be updated to reflect the new geometry. Similarly, the interchannel gap length and width must be updated. Other modifications to the model are also required, but in modifying only the fuel geometry hundreds of numbers must be successfully computed and transcribed into the appropriate VIPRE input file format.

By contrast, with the use of the Matlab-VIPRE Interface, a relatively short, high-level script is required. The main body of this script is given here:

```
%vipre_diam_vary.m
addpath('c:\matlab_sv13\work\research\vipre transient tools');
```

```
spreadsheet_filename = strcat(pwd,'\vipre_input_spreadsheet_updated_2.xls');
%open the excel file
[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
SPECIFIC_POWER = 25; %kW/kg
REFERENCE_SPECIFIC_POWER = 25;%kW/kg
P_D = 10.4/9;
diam_conversion_factor = 1/25.4; %25.4 mm/inch
geom_sheet = get(spread_sheet,'Sheets','Channels');
oper_sheet = get(spread_sheet,'Sheets','oper');
diameter_range = get(geom_sheet,'Range','e8','e8');
gt_diameter_range = get(geom_sheet,'Range','e13','e13');
pitch_range = get(geom_sheet,'Range','e20','e20');
wire_wrap_range = get(geom_sheet,'Range','e21','e21');
initial_flow_area = get_cell_value(geom_sheet,'c443');%in square inches
nominal_power = get_cell_value(oper_sheet,'e8'); %corresponds to a specific power of 25
kW/kg
nominal_flow_rate = get_cell_value(oper_sheet,'d8'); %lbm/sec
nominal_mass_flux = (nominal_flow_rate*3600*1e-6*(144)*(1/initial_flow_area)); %in
Mlbm/hr-ft^2

var_file_name = 'test_jac_var';
input_file_name = 'test_jac_input';

%input parameters in mm:
NOMINAL_PITCH = 10.4;
NOMINAL_ROD_DIAMETER = 9; %MM
NOMINAL_GT_DIAMETER = 9; %mm
NOMINAL_WIRE_WRAP = NOMINAL_PITCH - NOMINAL_ROD_DIAMETER;

%VARIATION = 0.5; %MM
NUM_STEPS = 7;
LOWER_ROD_DIAMETER = 8.5;
UPPER_ROD_DIAMETER = 9.725;
test_mdnbr = zeros(2,NUM_STEPS);
test_heat_flux = test_mdnbr;
test_eq_quality = test_mdnbr;
test_chf = test_mdnbr;
channel_heat_flux = zeros(48,NUM_STEPS);
test_pwrinp = zeros(NUM_STEPS,1);
test_gin = zeros(NUM_STEPS,1);
test_flow_rate = zeros(NUM_STEPS,1);
test_flow_area = zeros(NUM_STEPS,1);
diam_space_si = linspace(LOWER_ROD_DIAMETER,UPPER_ROD_DIAMETER,NUM_STEPS);
diam_space_brit = diam_conversion_factor .* diam_space_si;

for i = 1:NUM_STEPS
    set(diameter_range,'Value',diam_space_si(i)); %next diameter value
    set(gt_diameter_range,'Value',diam_space_si(i)); %keep the guide thimble moving too
  % ===== set the new fuel pitch to maintain P/D ===
    temp_pitch = diam_space_si(i)*P_D;
    set(pitch_range,'Value',temp_pitch);
    set(wire_wrap_range,'Value',temp_pitch - diam_space_si(i)); %set wire-wrap diameter
  %================================================
    invoke(spread_sheet,'Save');

    get_jacopo_excel_input(spreadsheet_filename,var_file_name,spread_sheet);

  % ======== set the new linear heat rate to maintain power density ==
    test_pwrinp(i) = (SPECIFIC_POWER/REFERENCE_SPECIFIC_POWER) * nominal_power *
(diam_space_si(i)/NOMINAL_ROD_DIAMETER)^2;
    change_input(var_file_name,'oper5.pwrinp',test_pwrinp(i));
  %=================================================================
  %=== set the mass flux so that delta T is constant ==============
    test_flow_area(i) = get_cell_value(geom_sheet,'c443');
    test_flow_rate(i) = (test_pwrinp(i)/nominal_power)*(nominal_flow_rate);
    test_gin(i) = test_flow_rate(i)/test_flow_area(i);
    change_input(var_file_name,'oper5.gin',test_flow_rate(i));
  %=================================================================
    generate_input_file(input_file_name,var_file_name);
    vipre_run_jon(input_file_name,var_file_name);
```

```
    %collect run data into workspace variables
    load channel_data;

    test_mdnbr(:,i) = corr_mdnbr;
end

set(diameter_range,'Value',NOMINAL_ROD_DIAMETER); %set the value back to what it should
be
set(gt_diameter_range,'Value',NOMINAL_GT_DIAMETER);
set(pitch_range,'Value',NOMINAL_PITCH);
set(wire_wrap_range,'Value',NOMINAL_WIRE_WRAP);
invoke(spread_sheet,'Save');%so the program doesn't ask me
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object
```

The desired relationship between MDNBR and rod diameter subject to the constraints of constant specific power and constant enthalpy rise is given in Figure 4.

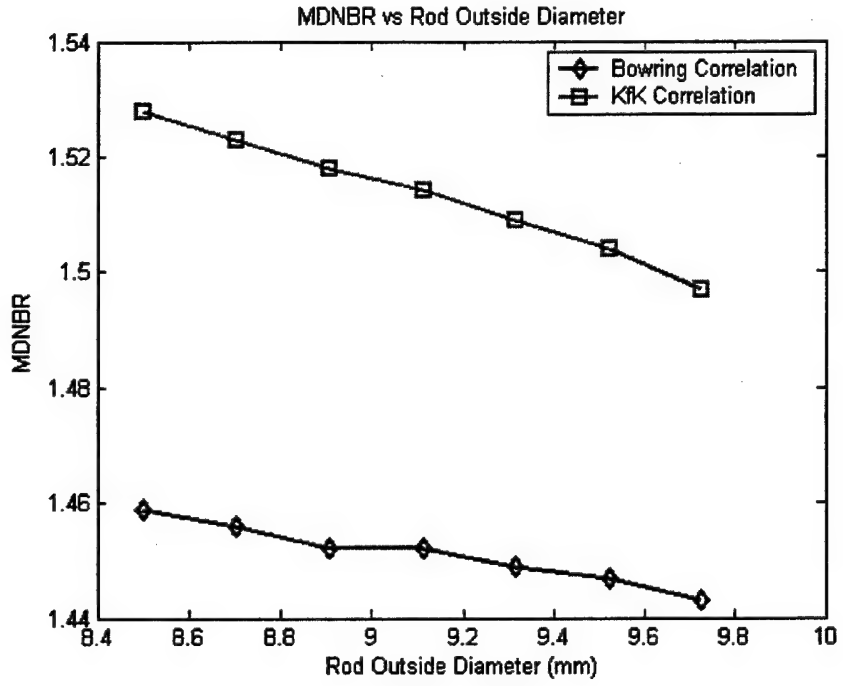


Figure 4: IRIS Tight Core MDNBR vs Rod Outside Diameter.

The main feature of this result is that the MDNBR is reduced as rod diameter is increased and correspondingly improved as the rod is made smaller. The DNBR limit established in the design of the IRIS Tight Core is 1.50. It can be seen from Figure 4 that, in light of this limit, the maximum permissible rod outside diameter is 9.6 mm.

Since mass flow rate is adjusted to maintain a constant enthalpy rise across the core for the different rod diameters, and the axial power shape is maintained constant, the

equilibrium quality of the coolant is constant for a given axial level as rod diameter is changed. Because of this, the change in MDNBR for varying rod diameters is expected to be dominated by changes in rod surface heat flux.

The rod heat flux is influenced by the changing rod diameter in two ways: by changes in the rod surface area, and changes in linear power for a given rod. The change in linear power with changes in rod diameter is shown directly in Equation 4. The surface heat flux for a rod with a given linear heat rate is:

Equation 7

$$q'' = \frac{q'}{\pi D_{rod}} \quad \text{where: } q'' \text{ is the heat flux}$$

It can be seen by inspection that, for a constant q', q'' is dereased as rod diameter is increased.

Combining Equation 7 with Equation 4,[v] the change in heat flux with changing diameters with a constant specific power is shown:

Equation 8

$$q''_{new} = \frac{q'_{new}}{\pi D_{rod\_new}} = \frac{q'_o D^2_{rod\_new} / D^2_{rod\_o}}{\pi D_{rod\_new}} \Rightarrow \frac{q''_{new}}{q''_o} = \frac{\dfrac{q'_o D^2_{rod\_new} / D^2_{rod\_o}}{\pi D_{rod\_new}}}{\dfrac{q'_o}{\pi D_{rod\_o}}} = \frac{D_{rod\_new}}{D_{rod\_o}}$$

This expectation that q'' will increase if the new rod diameter is larger than the old diameter and consequently the MDNBR will decrease is borne out in the VIPRE analysis results.

The purpose for this section was to demonstrate the power provided by the script interface. This task, while straight-forward conceptually,[w] would be tedious and time-consuming to perform by hand. In contrast, this analysis was conducted rather quickly

---

[v] Note that Equation 4, q' is given in terms of the fuel pellet diameter, where in Equation 7, the heat flux is given in terms of the fuel rod diameter which includes the fuel-clad gap, and the gap thickness. The fuel model used in this analysis sets the fuel rod diameter to be a constant multiple of the fuel pellet diameter. Since these equations ultimately involve the ratios of these quantities, fuel rod diameter and fuel pellet diameter can be used interchangeably provided they are used consistently.

[w] ...And therefore fairly easy to program...

with the script-interface and could easily be modified to perform a wide variety of other interesting analyses according to the user's need.

2.5. Extensions Of the Interface to Other Codes

The methodology used in this application for providing a scripted interface to VIPRE can be applied many other codes used within the Nuclear Engineering Department at MIT. This section outlines script-based interfaces generated for the fuel performance code FRAPCON and the plant simulation code RELAP.

There are two features in particular that facilitate the creation of a script-based interface for a given code. First, the code must normally utilize a file-based interface. While this is a common feature among legacy codes, most contemporary programs employ a GUI and the primary method for providing input data and invoking the execution of the program. Unless special provisions where made in the creation of the program,[x] there is no ability to provide input data, or initiate the analysis portion of the program externally to this GUI. This makes the incorporation of a script-based interface impossible.

Second, the source code[y] for the program should be available and in a form that eases source code modification.[z] While this is not essential, it is very helpful to be able

---

[x] There exist standard methods, when developing for a Windows environment, to provide functionality that allows one's program to run as an Activex Server. This would allow the script to possibly act as an Activex Client thereby making the desired behavior of automated modification and execution of portions of the program possible. An example of a program with this capability is Microsoft Excel – this is the reason why Matlab is able to acess and modify Excel spreadsheet values from within Matlab. This same provision exposes much of the functionality of Excel. For example, Excel workbook functions can be called and applied to data passed from Matlab, with the result returned to Matlab.

[y] It is understood that in the case that the original code developers are a company that intends to profit from the sale of the program, they may be reluctant to release the source code. I would briefly advocate here that it is possible to protect a company's substantial intellectual property investment in the code while still providing this functionality. Rarely is any serious intellectual investment required for the input and output routines within the source code and likewise, need not be rendered unavailable. The more sensitive data and computational routines could be packaged within libraries and thus protected.

[z] For a particularly complex source-code file structure, this can be essential. The codes VIPRE and FRAPCON were easily organized and compiled using Visual Fortran. On the other hand RELAP has a particularly complex code structure, it was necessary to obtain from a third party a form of the code arranged in a Visual Fortran workspace to allow compilation. In the UNIX operating system, it is common practice in cases where the source code is provided, to also provide standard configuration shell scripts and compilation files (such as Makefile and make.inc among others) to facilitate code maintenance and

to modify the source code for purposes of constructing new output files that contains only the specific data that is sought after. This relieves the designer of the script interface of the requirement to develop the logic structure required to parse a general output file for the specific data required.[aa]

In the following sub-sections, a description is given of the, generally more limited and incomplete, script-interface functionality that has been provided for FRAPCON and RELAP, and how it can be used in co-operation with the script interface created for VIPRE.

Frapcon

The Matlab-FRAPCON interface is designed in much the same way as the Matlab-VIPRE Interface. A schematic representation is show in Figure 5.

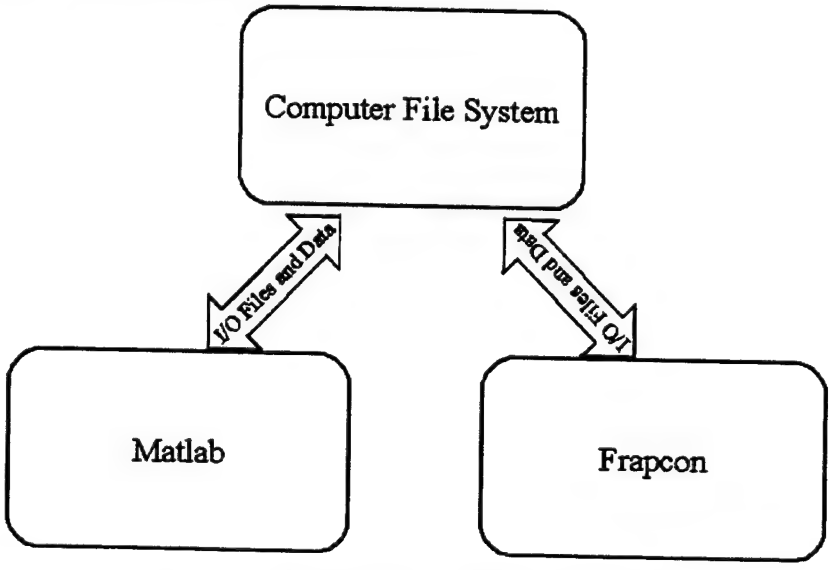


Figure 5: Schematic Representation of Matlab-FRAPCON Interface.

Unlike the interface developed for VIPRE, all of the data required to generate a valid input file for FRAPCON is stored in a relatively short script accessed directly by Matlab.

---

modification. If a script interface were to be created for a code in a variant of the UNIX operating system these files should also be available.

[aa] I.e. It is difficult in general to separate the numeric data from the accompanying explanatory text contained in an output file. If, on the other hand, the file is only numeric, with the numbers arranged in the format dictated by the interface designer, data parsing can be done simply and with much more generality.

Scripted Coupling of Codes for Multi-Physics Simulations

As discussed hitherto in this chapter, when the concept of providing a Matlab-based script interface to a code is brought to reality, many benefits are provided in the way of parametric analysis and optimization. Further benefits can be derived by providing the same scripted interface to more than one code. When this is done, the analytical capabilities of both codes may be combined in such a way as to improve the applicability or fidelity of the analysis.

As an example of this concept is the Matlab-VIPRE-FRAPCON Interface. A schematic representation of this is given in Figure 6.

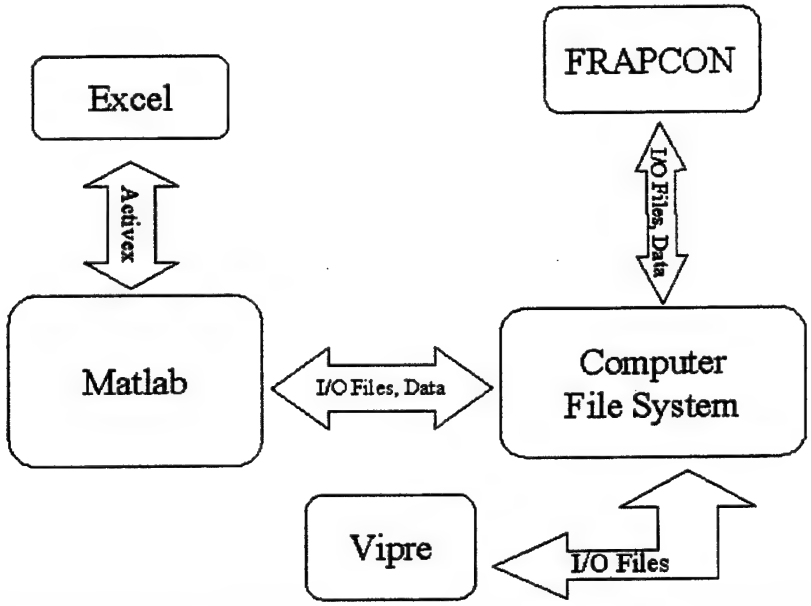


Figure 6: Schematic Representation of the Matlab-VIPRE-FRAPCON interface.

The figure is drawn to suggest that VIPRE and FRAPCON are somehow coupled together. This is true only in the loosest sense. It is the input parameters provided to each code and the output values provided by each code that separately coexist in the Matlab workspace context. Scripts can be written, that can, for example, look at the output of an appropriate FRAPCON analysis and incorporate these results into a VIPRE input file for the purposes of improving the accuracy of the VIPRE model.

An example application of this approach would be for modeling the changes in T/H behavior for a very long cycle length (VLCL) core. In particular, for VLCL core designs

that are very sensitive to minor perturbations in the fuel geometry. An example of a design with these properties is the IRIS Tight core described in section 2.5.

This is a core where the pitch-to-diameter ratio was designed to be nearly as tight as possible. For a fixed fuel rod pitch, even small changes in the rod outside diameter have a dramatic impact on critical design parameters such as MDNBR, core pressure drop, and coolant flow velocity. A demonstration of this sensitivity is given in Figure 7. A problem lies in the fact that, for a VLCL core, fuel dimensional changes are very likely to occur, and it would be useful to take these changes into account in a quantitative way, if possible.

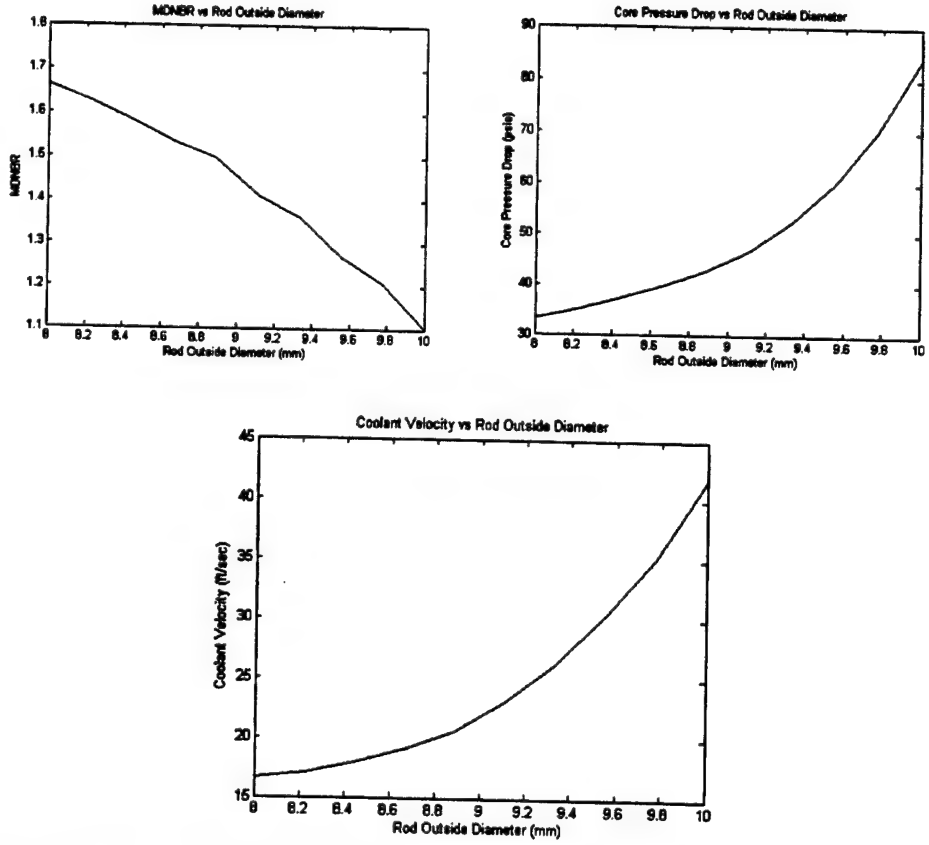


Figure 7: IRIS Tight - Effect of Rod Diameter Variation with Constant Rod Pitch, Linear Power and Mass Flux.

For VLCL cores, physical phenomena occurring within the core such as cladding corrosion, fuel swelling, fission product gas release and cladding creep act to change the rod outside diameter, and therefore impact the T/H conditions in the core. FRAPCON is exactly the sort of tool required to quantify the cumulative impact that all of these

processes have on the performance of the fuel over the life of the core. The use of an integrated script interface to both VIPRE and FRAPCON, allows for automated transfer of data resulting from FRAPCON analysis to VIPRE input files to update the T/H model to account for the changing fuel conditions.

Chapter 3 IRIS Open Core Thermal Hydraulic Analysis

3.1. Introduction

Thermal hydraulic (T/H) analysis plays a critical role in determining the acceptability of any prospective reactor core design. This analysis must be made as an integral component of the overall core design effort that will include other important analysis such as:

- Core neutronics design: Is the reactor critical? What is the core reactivity lifetime? What is the core power distribution (axial and radial)? Are the reactivity coefficients within acceptable limits?
- Steam Generator (S/G) and balance of plant design: How much surface area does the S/G require? What is the minimum S/G tube thickness allowed? What will be the flowrate of secondary systems fluids?
- Containment design: How does one ensure that the containment will not fail in the event of a large loss of coolant accident (LOCA)?

The above list is not exhaustive, but should give a good feel for the context in which the T/H analysis must exist. For the core T/H analysis, the main questions are: Can I remove the heat generated in the core while maintaining the fuel clad at an acceptable temperature during steady state operations? Will the fuel centerline temperature be acceptable under the same conditions? Can I prevent fuel or cladding damage in the event of certain foreseeable accidents? How do system design and behavioral uncertainties impact the answers to the above questions?

Not all of these questions are addressed in this chapter. In this chapter, the philosophy, methodology and results will be provided for the IRIS Open Core steady

state and transient core T/H analysis. The transients to be considered are all specific instances of a more general category of accident referred to as the Loss of Flow Accident (LOFA). In particular the casualties are:

- Locked-Rotor/Shaft Shear (LRSS): In this accident, one reactor coolant pump (RCP) is incapacitated by either the seizure of the pump rotor or loss of mechanical integrity of the pump motor shaft. In either case, the pump loses it's ability to circulate cooling water to the core.
- Partial LOFA (PLOFA): In this accident 4 of the 8 RCPs stop operating for unspecified reasons.
- Complete LOFA (CLOFA): In this accident, all 8 RCPs stop operating for unspecified reasons.

These accidents will then be analyzed in such a way as to include in the analysis the effects of core operational parameter and design uncertainty using three different methodologies of varying degrees of conservatism.

3.2. Thermal Design In Practice

The goal of T/H analysis is to ensure that the core can be safely cooled over the design life under normal and accident conditions. This analysis must make provisions for the uncertainty inherent in engineering design and construction, as well as uncertainty in core thermal performance over the core lifetime. As allowances are made for the occurrence of operational transients and for uncertainties in and changes of core behavior over core life, the allowed core power must be reduced. This is illustrated in Figure 8.

THERMAL ANALYSIS IN PRACTICE

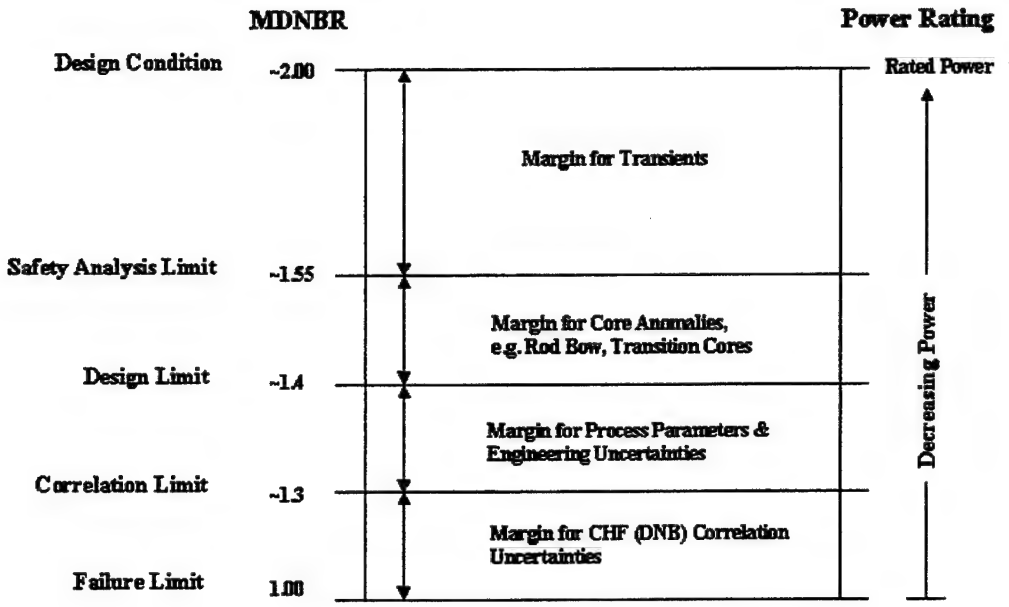


Figure 8: Thermal Analysis in Practice.

In the absence of any uncertainty, the core Power Rating could be set such that the MDNBR is at the Failure Limit, defined in this context to be the condition where MDNBR = 1.

Unfortunately, the correlations used to determine MDNBR for a known set of core conditions is subject to error. To account for this uncertainty, it is required that the calculated MDNBR be sufficiently greater than one so that the analyst can have reasonable confidence that the *actual* MDNBR is greater than one.[a] One way that this can be accomplished is, by using the statistical properties of the test data used to formulate the DNB correlation, to increase the limit MDNBR by an amount so that there is a 95% probability with 95% confidence[b] that DNB has not occurred. This value is termed the Correlation Limit, and Table 5 is a short list of these limits for a few correlations. This effect is shown on the left side of Figure 8, and implies a reduction in the core Power Rating (shown on the right side of Figure 8). For those correlations listed where this Correlation Limit DNBR has not been established: EPRI, Bowring, MacBeth

[a] I.e. What does the calculated MDNBR have to be to ensure that DNB has not occurred in the core.
[b] Colloquially referred to as: "95/95 confidence".

and AECL-UO Look-Up Table, a value of 1.30 was used. As shown in Figure 8, the Correlation Limit DNBR is given as 1.30.

Correlation	Correlation Limit DNBR
WRB-1	1.17
W-3L	1.30
B&W-2	1.32
CE-1	1.13

Table 5: Correlation Limit for Some DNB Correlations (Source: reference [11]).

In a similar fashion, the core as initially designed has properties that are subject to uncertainty. Examples include: fuel rod diameter, fuel cladding thickness, fuel enrichment, core flow rate, and coolant temperature. Because of these uncertainties, at a given power level, portions of the core may be closer to thermal limits than that which is computed otherwise. To account for these uncertainties, the allowed Power Rating must be reduced and, equivalently, the MDNBR limit must be increased. This is referred to as the Design Limit MDNBR. The numerical value of the Design Limit MDNBR is assumed to be approximately 1.4 in Figure 8, but the actual numerical value used is dependent upon the specific core and the specific CHF correlation being used.

Over the life of the core, due to material degradation mechanisms or, in the case of transition cores, changes are made to core design[c] from use of mixed fuel assembly designs. These changes result in either or both T/H degradation and an increase in design uncertainty that must be further accommodated by lowering the Power Rating, and thus increasing the nominal steady state MDNBR. For this example an additional 0.15 is added to the Design Limit MDNBR to allow for this uncertainty leaving the Safety Analysis limit MDNBR at approximately 1.55.

[c] More specifically – transitional changes in core design. An example of this would be the gradual upgrade of fuel assemblies over several fuel batches.

Finally, core transient behavior must be taken into consideration. To account for the T/H degradation caused by plant operational transients, additional thermal margin must be added and the Power Rating reduced. For this example, the value of 2.0 was assumed to be sufficient. When all uncertainties and transients are taken into account, the Design Condition power and MDNBR is obtained.

It cannot be overstressed that the exact allowances that must be made to the core power limit or MDNBR limit are strictly core (or, fuel assembly) and DNBcorrelation dependent. The typical values that are stated in this section are based on a PWR[d] using a correlation such as W-3L. The results would be completely different, for example, if the EPRI correlation were used instead.[e]

3.3. Thermal Hydraulic Analysis Methodology

This is a chapter presenting T/H analysis methodology not design or optimization results. Nearly all of the design decisions for the IRIS Open Core had been made previous to the analysis described here. Core design parameters such as fuel rod thickness, fuel rod and assembly pitch, coolant mass flow rate and core thermal power have already been determined by IRIS consortium members from around the world. A summary of these activities can be found in reference [12]. Given this status, no consideration was given to further parametric analysis or optimization studies. The design is merely analyzed in its present state.

The tasks to be completed for this analysis are enumerated in this section. The following sections provide a more detailed description of each task. The tasks are:

- Collect data required to model the IRIS Open Core using an accepted T/H analysis code. For this project the subchannel analysis code named **V**ersatile **I**nternals and Component **P**rogram for **R**eactors: **E**PRI (VIPRE) was selected.
- Evaluate core T/H performance at steady state. Ensure that the minimum departure from nucleate boiling ratio (MDNBR) is sufficiently high so as to

[d] The thermal limits in question (in this case, MDNBR) are not, for example, applicable to a liquid metal, or gas cooled reactor.

[e] The response of various DNB correlations to changes in linear power is compared in section 3.5 of this thesis. In section 3.5, it is shown that correlations such as EPRI and Bowring, respond much differently to changes in linear power than W-3L or B&W-2.

provide adequate margin during transient analysis. Also, ensure that fuel clad and centerline temperatures are sufficiently low.

- Develop a set of input data for a plant simulation model. For this project the code RELAP was used. This code was used to find macroscopic plant parameters during the course of a postulated transient such as: core inlet temperature, system pressure, core thermal and neutronics power, and coolant flow rate. This information was then used as input data to the VIPRE code for transient simulation
- Perform T/H transient analysis using VIPRE. Ensure that the lowest MDNBR attained during the transient is above the required limits.
- Perform T/H sensitivity analysis (using VIPRE). Determine the percentage change of MDNBR for a given percentage change in a given input variable. This analysis provides a key input to the uncertainty analysis procedure
- Perform T/H uncertainty analysis. This task is meant to ensure (with a required level of confidence) that the core T/H conditions will be acceptable even in the presence of design uncertainty.

The tasks listed above do not represent an exhaustive list of all tasks required for a full T/H analysis. Additional analysis would be required for the following:

- Establish an envelope of safe steady state operations – In principal this envelope would determine the initial conditions from which accident analysis would begin. The worst-case initial conditions would represent the initial conditions for a given transient[f] that produces the most damaging result. This set of conditions would be used for the transient plant simulation. Researchers at Westinghouse have initiated transient analysis independently using nominal parameters for initial plant conditions.
- Determine the set points for plant protective features. For example this analysis would be used to determine at what power level a scram signal should be initiated.

These analyses were considered to be outside the scope of this work.

3.4. IRIS Open Core Model Description

Engineers at Westinghouse Electric Co. Science and Technology Division formulated the IRIS Open core model. In so doing, many modeling decisions were made based on standard Westinghouse T/H design procedures. The basic model parameters are described here.

[f] Not all transients would have identical "worst-case" initial conditions. For some transients such as LOCA, high power and high temperature, with an extended 'thermal history' of recent long-term high power operations would represent the worst-case conditions. For some reactivity induced accidents such as a sudden rod withdrawal – low initial power, low temperature would be limiting.

The whole core is represented by a 1/8th core section that is consistent with the geometrical and physical symmetry present in a typical PWR. The original core called for a Westinghouse standard 15x15 fuel assembly design. It was determined during steady state and transient core T/H analysis that the 15x15 design was not delivering the thermal performance hoped for.[g] The 17x17 fuel assembly was chosen as an alternative. The assembly linear heat rate is the same as for the 15x15, but the per-rod linear heat rate is lower simply because there is a larger number of fuel rods in the 17x17 array. Despite the smaller rod diameter, the increased total surface area translated into a 6 percent reduction in heat flux in the 17x17 fuel rods.[h] This was found to be sufficient cause to change the 17x17 fuel assembly design.

Generic parameters were used to account for grid and mixing coefficients, turbulent and laminar frictional coefficients, and correlation selections for various thermal hydraulic flow and heat transfer regimes. A detailed discussion of these decisions can be found in reference [12] chapter 5. The 1/8th core subchannel nodalization scheme and pertinent design information is presented in Figure 9.

Parameter	Value
Core Thermal Power	1000 MW
Core Average Linear Heat Rate	3.04 kW/ft
System Pressure	15.58 MPa (2260 psia)
Total Core Flow Rate	36,000 kg/s (1.235 Mlbm/ft^2-hr)
Core Average Coolant Inlet Temperature	292 °C (557.4 °F)

[g] The MDNBR was too low.
[h] The fuel centerline temperature also was considerably lower for the 17x17, though this is not a limiting condition for the accidents considered in this analysis.

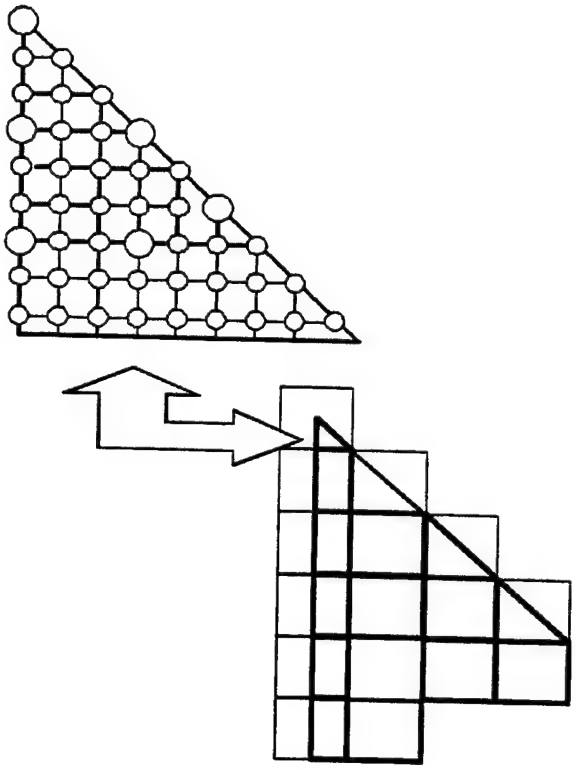


Figure 9: IRIS Open Core subchannel nodalization and design information

3.5. Steady State Thermal Hydraulic Analysis

The limiting parameter for the steady state T/H analysis for this core is the MDNBR.[i] This parameter requires the use of an empirical correlation. The choice of which correlation to use is of critical importance to the validity of the T/H analysis. For this analysis, several CHF correlations were used and compared to one another.

The physical processes that underlie the phenomenon of CHF have been studied thoroughly in both theory and experiment. Summaries of this work can be found in references [13]-[17], though the body of literature on this subject is enormous. Despite the attention given to CHF, the phenomenon has resisted all efforts at developing a predictive model based completely on theoretical 'first-principles'. The process whereby

[i] In a more general thermal hydraulic design, several parameters can potentially be design limiting. Some common examples include: fuel centerline temperature, core pressure drop, coolant velocity and core temperature rise.

the fluid, flowing along the channel walls is replaced by a blanket of steam bubbles is difficult to explain concisely, and extraordinarily complex to characterize exactly; so dependent is it on the exact physical configuration under consideration. Many natural phenomena of interest are like this – as examples: fully developed turbulence and reaction chemistry.[18] Despite this, we can model some processes quite well because they are regulated by higher-level physical laws. For example, the laws of thermodynamics, the laws of plasticity, and the laws of hydrodynamics make meaningful simulation possible – though they are built upon the lower-level laws of quantum mechanics from which direct simulation would be impossible.[j]

The choice as to which correlation is 'most acceptable' is one that must be made using the experience and judgment of the analyst. In the final analysis, it must be proven by experiment that CHF will occur, for a given fuel assembly design, in a way that is accurately described by the CHF correlation that is to be used. Alternatively, and perhaps more likely, a new CHF correlation could be developed[k] that takes maximum advantage of design features of the fuel assembly.[l]

The fuel assembly to be used in this core can be most accurately and reliably analyzed using the WRB-2 correlation developed by Westinghouse. This correlation takes into account the carefully designed, fabricated and installed support and mixing grids provided for improved heat transfer within the fuel assembly. Additionally, this correlation has been extensively tested and validated so as to have proven capability for the particular fuel assembly selected. Unfortunately, the WRB-2 correlation is proprietary and cannot be used in this analysis.

As a substitute to the WRB-2, the W-3L, EPRI, Bowring, Babcock & Wilcox-2 (B&W-2), MacBeth and the Atomic Energy of Canada Ltd – University of Ottawa (AECL-UO) Look-up Table are used for comparison.

---

[j] All students and practitioners in fields requiring comuter simulations should read and memorize reference [18].

[k] Or, more accurately, a slight variation of an existing correlation would be developed.

[l] Much effort is expended in the way of improving the thermal performance of the fuel assemblies. This is a key activity for a company that is in the business of selling fuel assemblies to nuclear power plant operators. The design improvements would be of little use if they were not captured by CHF correlations.

For the analysis presented in this thesis, the CHF correlations will be of particular interest. These correlations, along with their data ranges of are presented in Table 6.

Correlation	Pressure (psia)	Mass Flux $Mlbm/ft^2-hr$	Quality	Heated Length (ft)	Hydraulic Diameter (in)	Geometry Type
IRIS Open	2259.4	1.235	-0.3-0.02[m]	14.0	0.529	Rod bundles
B&W-2	2000 – 4000	0.75-4.0	-0.3-0.2	6.0	0.2-0.5	Rod bundles
W-3L	1460 – 2460	1.96-3.68	-0.15 – 0.15	8.0 – 14.0	0.2 – 0.7	Rod bundles
MacBeth	15 – 3000	0.0073 – 13.7	*	0.08 – 12	0.04 – 1.475	Round tubes
Bowring	90 – 2250	0.04 – 3.0	*	5 – 15	0.03 – 14	Inlet subcooling
EPRI-I	200 – 2450	0.2 – 4.5	-0.25 – 0.75	2.5 – 15	*	PWR, BWR (Matrix subchannels only)
AECL-UO Look-Up[n] (Ref [19])	14.5 — 2900	0 – 5.53	-0.5 – 1.0	See note	See Note	See Note

* These values are not available

Table 6: Data Ranges for Critical Heat Flux Correlations (Reference [2]).

Direct comparison of different CHF correlations is a subtle and perhaps unappreciated business. Figure 10 is provided as an illustration of this complexity. At the nominal design power of 3.04 kW/ft, the MDNBR, as determined by the six

[m] Corresponds to inlet and outlet equilibrium quality for nominal steady-state conditions.
[n] The AECL-UO Look-Up table uses correction factors to take into account the effect of heated length, hydraulic diameter and the geometry type -- a bundle correction factor is used.

different correlations, ranges from the lowest: EPRI – MDNBR = 1.667, to the highest: MacBeth – MDNBR=5.078. Because of the significant disagreement of the MacBeth correlation with all others presented, the MacBeth correlation will not be used when considering the T/H acceptability of the IRIS core. It will continue to be considered, however, in interest in observing the behavior of this correlation in comparison to the others.

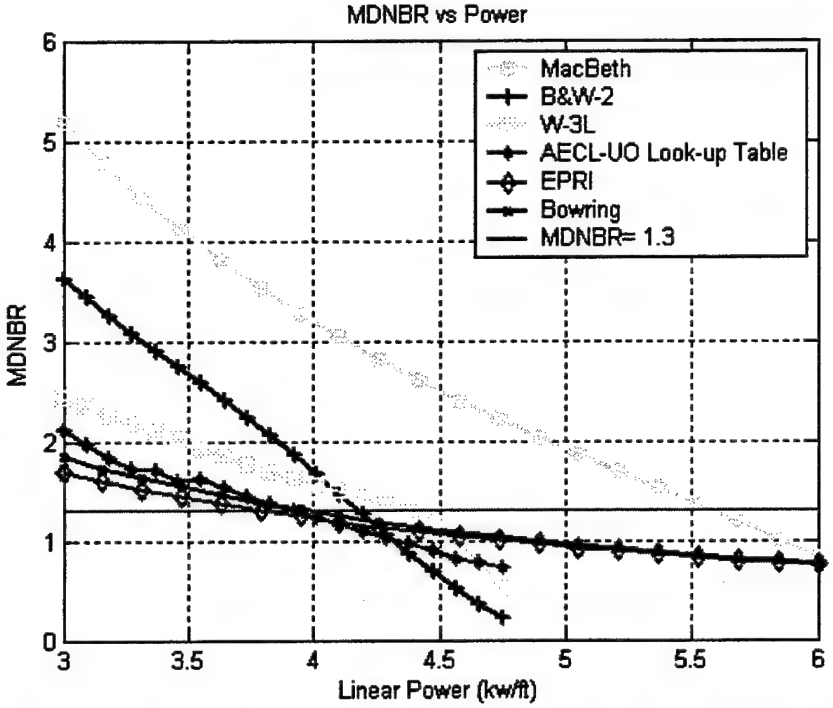


Figure 10: Comparison of Various CHF Correlations for IRIS Open Core. Cosine Power Shape, peak = 1.55.

While a broad range of MDNBRs is indicated for the core at nominal power, the powers for which the correlations predict the occurrence of CHF are relatively closely spaced. This illustrates the difference between DNBR, as a ratio of local heat flux to that which would cause CHF, and the Critical Power Ratio (CPR) which is the ratio of the total power (includes channel power history) to that which would cause CHF. Based on the nominal power, the CPR for the six correlations is provided in Table 7 and a plot of MDNBR versus CPR is given in Figure 11. It should be noted, that this analysis does not take into consideration any inaccuracy in the CHF correlation itself.

Correlation	MDNBR at Nominal Linear Power of $3.04\left(kW/ft\right)$	Linear Power for MDNBR = 1.0 $\left(kW/ft\right)$	CPR at Nominal linear power of $3.04\left(kW/ft\right)$
EPRI	1.692	4.734	1.577
AECL-UO	2.111	4.336	1.445
Bowring	1.843	4.844	1.615
B&W-2	3.631	4.340	1.447
W-3L	2.433	4.518	1.506
MacBeth	5.179	5.908	1.969

Table 7: IRIS OPEN CPR for various correlations.

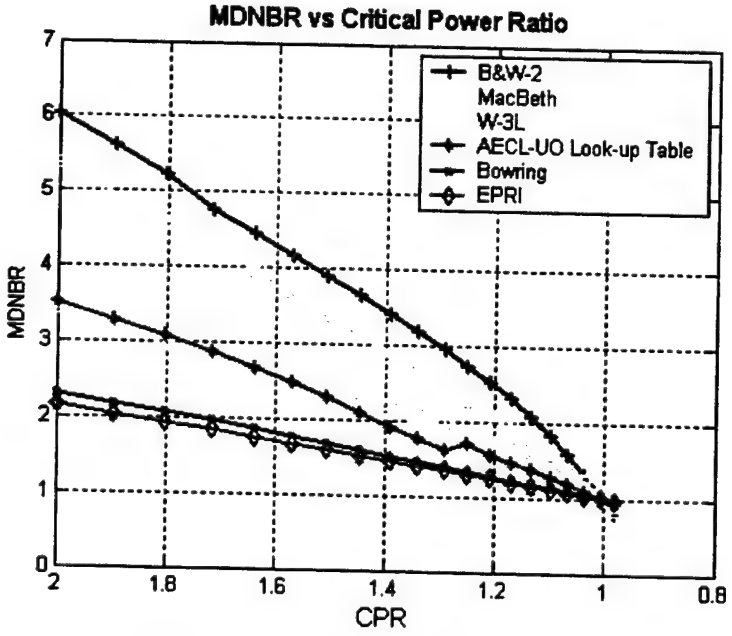


Figure 11: IRIS Open MDNBR vs CPR

The range of CPRs given in column 3 of Table 7 is much less wide than the MDNBRs provided in the first column of the same table. This is illustrative of the common misinterpretation of the reported DNBR as a measure of power margin at conditions less than the critical condition. For example, given only the DNBR, one could then assume that, while the EPRI correlation indicates only a modest margin to thermal

limits with which to absorb the effects of uncertainties and transients, the W-3L is much more optimistic and predicts more margin. The CPR analysis indicates that the two correlations make relatively similar predictions (ignoring uncertainties) concerning the power at which CHF will occur.

The reason for this behavior has its roots in the formulation of the different CHF correlations and is discussed in detail in reference [16]. Correlations such as EPRI and Bowring, have the functional form of:

Equation 9

$$q''_{CHF} = f\left(G, x_{in}\left(or\ h_{in}\right), p, D_h, z\left(or\ \frac{x - x_{in}}{q''}\right)\right)$$

Where:

q''_{CHF}	=Computed Critical Heat Flux
G	= Mass Flux
x_{in}	= Inlet Quality
h_{in}	= Inlet Enthalpy
p	= pressure
D_h	=Hydraulic diameter
z	=Axial height

For these correlations, the local quality is eliminated by incorporating the heat balance along the channel up to the point *z* of interest. For this reason, these correlations result in a computed CHF that is fairly constant despite changes in core power. This is manifested in Figure 11 as a line with a slope equal to negative one. Correlations such as W-3L and B&W-2 use only local channel conditions to determine the value of CHF and are of the form:

Equation 10

$q''_{CHF} = f(G, x, p, D_h)$ or, if heated length, L, is taken into account:

$$q''_{CHF} = f(G, x, p, D_h, L)$$

For correlations such as these, computed CHF increases when channel power decreases, causing MDNBR to increase faster than CPR when linear power is reduced. This is manifested as a more negative slope in Figure 11 as is shown. In this thesis, as will be pursued further in the Sensitivity Analysis section, the different responses of a

DNB correlation to increases in power will be manifest as an increase in power sensitivity.

3.6. Transient Thermal Hydraulic Analysis

The steady state thermal hydraulic analysis is a valuable tool in many senses, but for the purposes of fully determining the T/H acceptability of a core design, transient analysis is crucial. Reactor licensing rules require that the plant be able to undergo all Condition I and Condition II transients without sustaining fuel damage. The definitions of these transients are[20]:

- Condition I: These occurrences are operations that are expected frequently or regularly in the course of power operation, refueling, maintenance, or maneuvering of the plant; they shall be accommodated with margin between any plant parameter and the value of that parameter which would require either automatic or manual protective action.
- Condition II: These occurrences include incidents, any one of which may occur during a calendar year for a particular plant – examples include inadvertent control rod withdrawal and loss of a single reactor coolant pump. Condition II incidents shall be accommodated with, at most, a shutdown of the reactor with the plant capable of returning to operation after corrective action; any release of radioactivity shall be in conformance with paragraph 20.1 of 10CFR20.

The modeling and simulation of these transients is a task of significant magnitude. For this task, the plant simulation code RELAP was used. This code combines the detailed hydraulic and heat transfer characteristics of the entire reactor plant° along with the neutron kinetic behavior of the nuclear core. This analysis benefited greatly from the contribution of IRIS Consortium partners worldwide who created the RELAP model of the IRIS reactor plant.

While the RELAP code incorporates physical models of sufficient detail to provide reasonable fidelity with regard to macroscopic system behavior, the code in and of itself is not endowed with the models required to determine the detailed T/H conditions within the core. In particular no determination of the proximity to CHF is made. This task is relegated to VIPRE. RELAP does, however, provide the necessary time-dependent data for:

° Including the Steam Generating system and BOP.

- Average core inlet temperature
- System pressure
- Core thermal power
- Core coolant mass flow rate

This information is then provided to VIPRE for the purposes of the transient T/H calculation.

Complete Loss of Flow Accident

The CLOFA was modeled with the IRIS RELAP model, providing input to VIPRE as previously discussed. The general transient consisted of all 8 RCPs simultaneously losing power, followed 1.5 seconds later by a full reactor SCRAM. The time-dependent behavior of the plant during the CLOFA transient as computed by RELAP is shown in Figure 12.

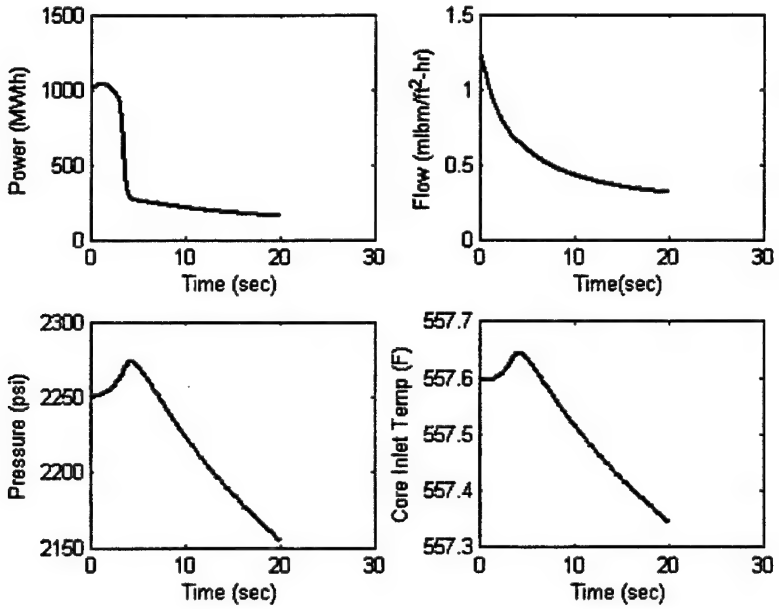


Figure 12: IRIS OPEN CLOFA transient profile.

Using the inputs provided by RELAP,[p] DNBR was computed for all modeled channels, for all time steps using the W-3L, Bowring, EPRI, B&W-2 and Macbeth

[p] The RELAP results used in this analysis are preliminary in nature. The RELAP model of the IRIS Open core is subject to nearly continuous revision and improvement. Since the focus of this work is both

correlations. The W-3L correlation exhibited poor performance as mass flux decreased. This is not surprising, as the low mass-flux is outside the range of applicability for the W-3L correlation. The CLOFA was modeled with three cases corresponding to possible coast down rates of the RCPs. The results of running VIPRE with the RELAP data for the nominal case are provided in Figure 13.

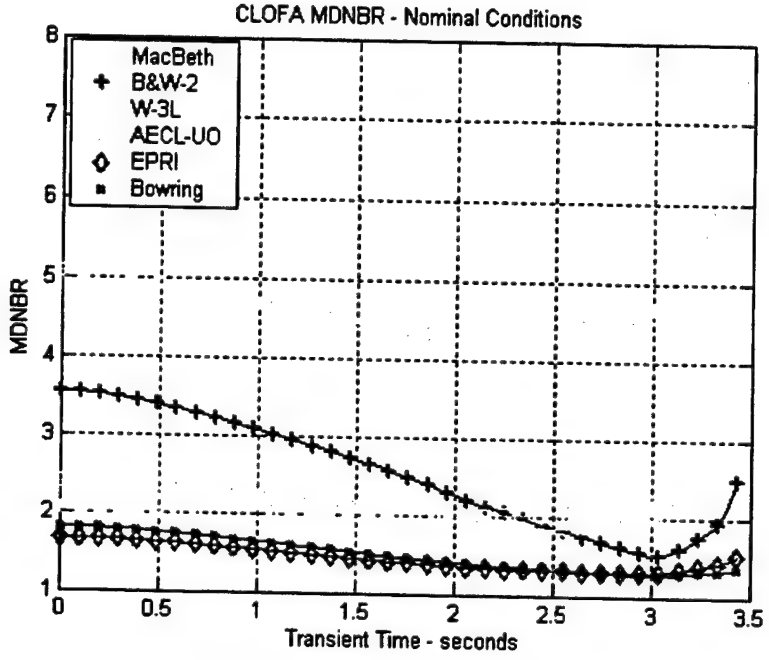


Figure 13: Nominal Conditions.

The coast-down rates are characterized by the term β which has dimensions (1/second) and is used in the following equation describing pump RPM as a function of time following pump trip:

$$Q(t) = Q_o\left(\frac{1}{1+\beta \cdot t}\right), \text{ where:}$$

Q(t) is pump speed as a function of time.

---

methodology and results, and because *some* version of the RELAP results had to be used (as opposed to continually repeating the calculations as each improvement in the RELAP model is implemented, these RELAP results are considered adequate for this analysis.

Q_o is the initial pump speed

and t is time from pump trip, in seconds.

If β is given a large magnitude, the model of the pump will indicate faster coast-down. The reverse is true if β is smaller. The nominal value of β is 0.33, the values of 0.21 and 0.5 were chosen as a reasonable upper and lower bound. The VIPRE-generated results for both cases are provided in Figure 14 and Figure 15.

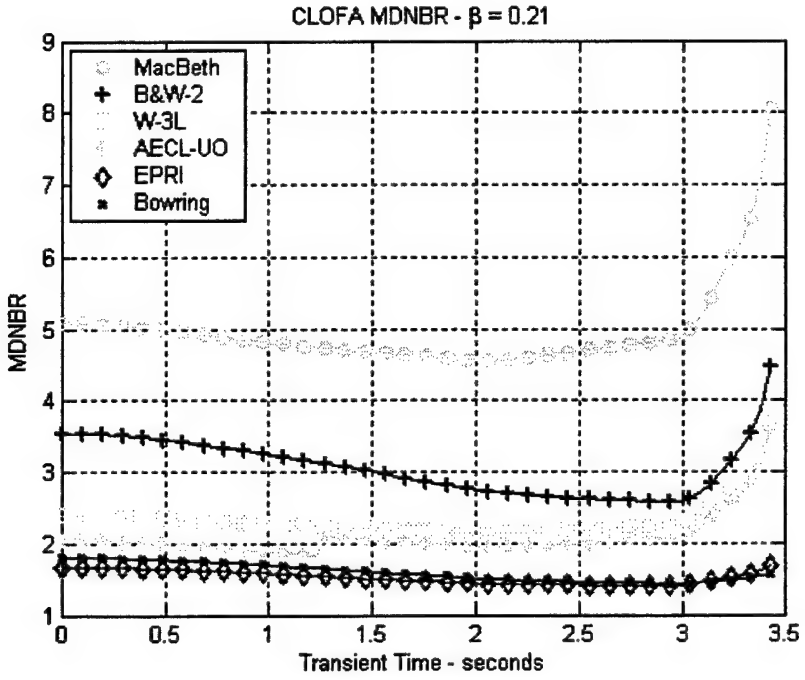


Figure 14: Slow Coast Down

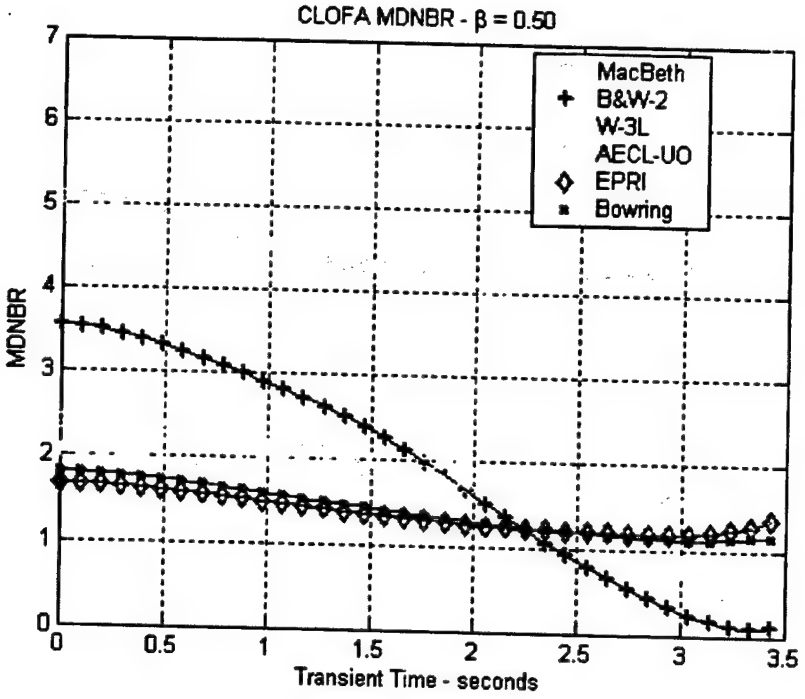


Figure 15:Fast Coast-Down

More recent analysis has demonstrated that β can be expected to be significantly lower than 0.33. (reference [21])

Partial Loss of Flow Accident 4 of 8 Reactor Coolant Pumps

This casualty was modeled in a similar manner to the CLOFA with the exception that only 4 of the 8 RCPs were halted for the transient. The reactor receives a Loop Low Flow trip signal when flow reaches 87% of nominal which occurs approximately 0.9 seconds into the transient. The time-dependent behavior of the plant during the PLOFA transient as computed by RELAP is provided in Figure 16.

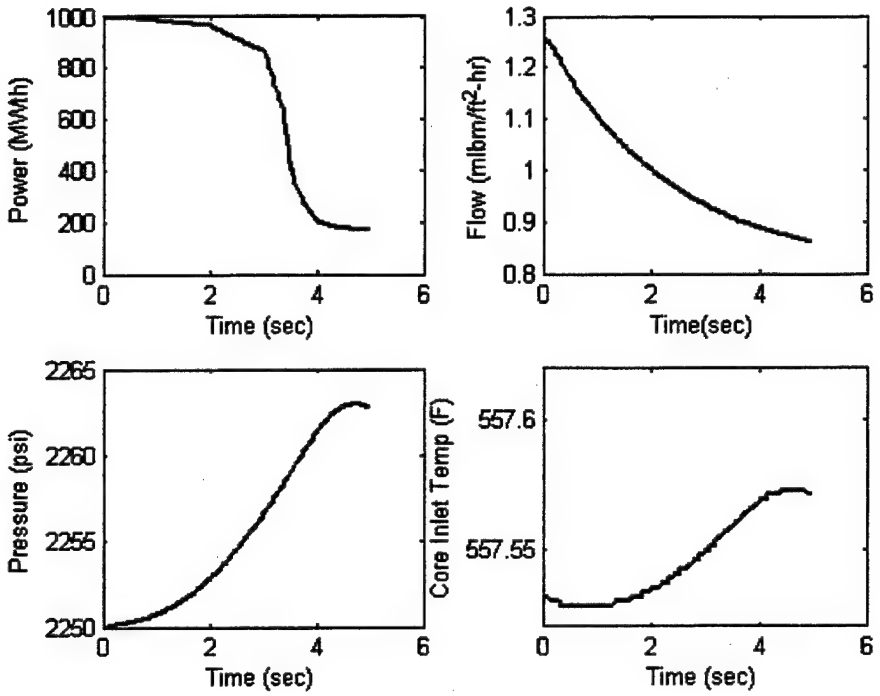


Figure 16: IRIS OPEN PLOFA Transient Profile.

The VIPRE results obtained for the 17x17 core for the transient conditions of Figure 16 are presented in Figure 17.

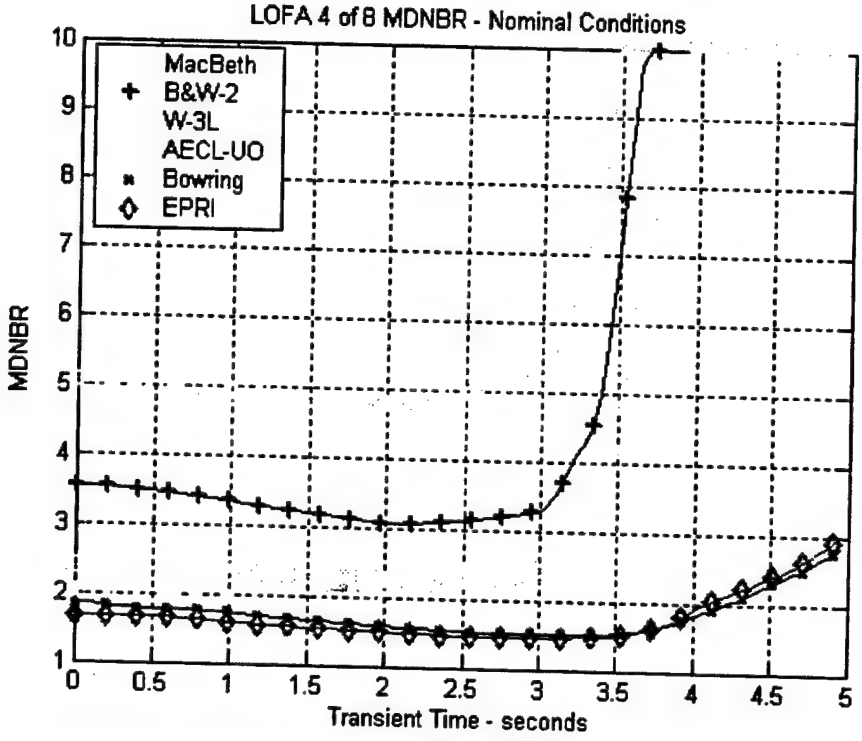


Figure 17: Partial Loss of Flow Accident 4 of 8 Nominal Conditions.

One might notice from this result that, after approximately 3.5 seconds, the MacBeth, B&W-2, and W-3L correlation MDNBRs become steady at a value of 10. This is a result of the way that VIPRE reports DNBR values. If a correlation predicts a DNBR greater than 10, the code simply returns the value 10. If the correlation predicts, due to the numeric behavior of the DNB correlation a negative value of DNBR,[9] then the code simply prints out a warning to indicate such a condition.

Locked Rotor/Shaft Shear

This transient was modeled in a similar fashion as the PLOFA, but in this case only one pump was stopped. The time-dependent behavior of the plant during the LR/SS casualty is illustrated in Figure 18.

[9] This condition often occurs with the W-3L correlation for conditions of low mass flux. Communication with Westinghouse Scientists indicate that this behavior is typical of Westinghouse Correlations.

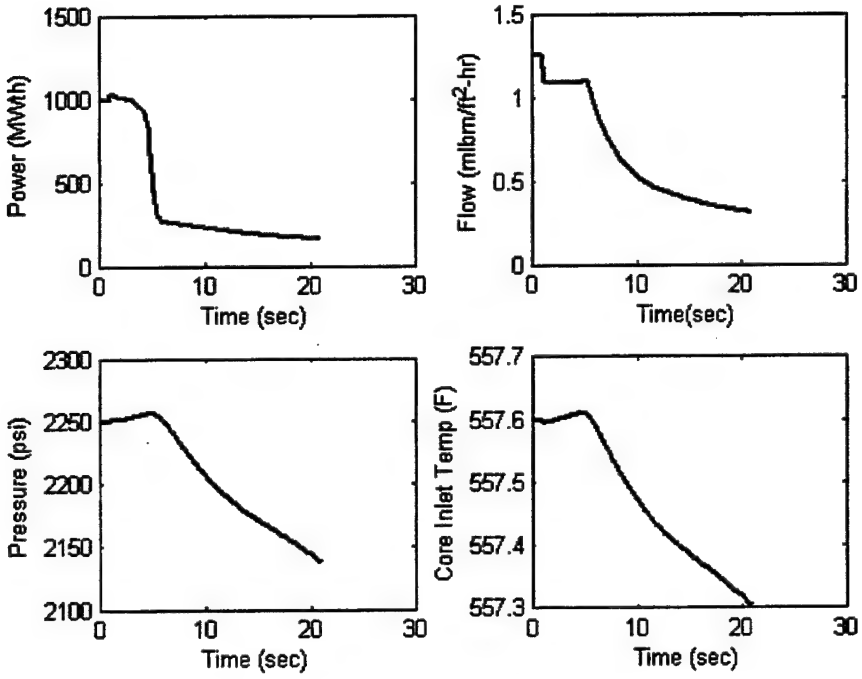


Figure 18: IRIS OPEN LR/SS Transient Profile.

The results for the 17x17 case are provided in Figure 19.

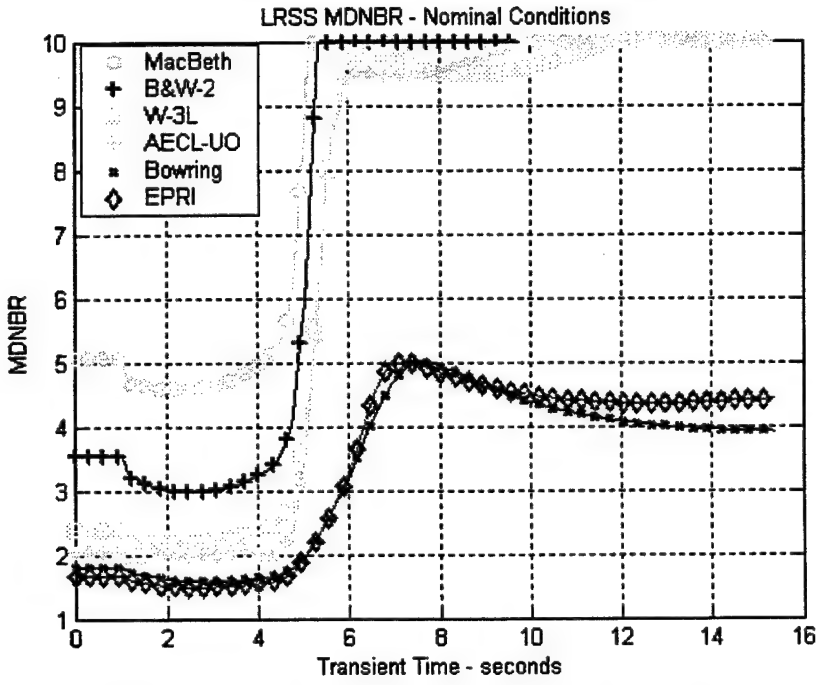


Figure 19: LR/SS transient results - nominal conditions.

In summary, for the plant initially at nominal conditions, the MDNBR results for the preceding transients are provided in Table 8.

	MacBeth	B&W-2	W-3L	Bowring	EPRI	AECL-UO
LR/SS	4.588	2.974	2.180	1.578	1.495	1.763
PLOFA	4.753	3.055	2.238	1.589	1.516	1.848
CLOFA[r]	4.308	1.527	1.993	1.276	1.296	1.828

Table 8: Summary of Nominal Condition Transient Results.

3.7. Sensitivity Analysis

After having completed the steady state and transient analysis with the core initial conditions set to their nominal values, it is desirable to know quantitatively the impact on MDNBR that should be expected for a given perturbation in the plant conditions. For the purposes of thermal hydraulic analysis, knowing this relationship provides relevant insight not only of plant behavior, but also of the relative behavior of various CHF correlations.

The knowledge gained in the sensitivity analysis is of critical importance for performing uncertainty analysis. The various methods employed for uncertainty analysis will be discussed in detail in the next section. One procedure, the Westinghouse Improved Thermal Design Procedure (ITDP), which is given a detailed treatment in reference [22], uses sensitivity factors as the centerpiece of the uncertainty analysis. Even if one is not using the ITDP as their main uncertainty analysis methodology, the sensitivity analysis can give quantitative indications of which plant parameter contributes most to uncertainty in the MDNBR for any given set of conditions. This can then

[r] Only base-case is considered here.

indicate to plant designers ways in which overall plant uncertainty can most economically be reduced, leading to improved plant thermal margins.

The sensitivity is dependent upon both the parameter under consideration and the DNB correlation in use. Sensitivities were obtained for Inlet Temperature, System Pressure, Mass Flux, Linear Power, and Engineering Enthalpy Rise Hot Channel Factor[s] ($F_{\Delta H}^{E}$) for the W-3L, B&W-2, MacBeth, Bowring, EPRI and AECL-UO Look-up Table DNB correlations using two different methodologies.

For the first procedure, ten data points were chosen over a selected range of interest for each parameter. The range for each parameter is shown in Table 9 as the 'Minimum' and 'Maximum' value for each parameter. At the ten data points within this range, ten steady state VIPRE runs were conducted with the parameter of interest varied over a ± 1% range about the given data point.[t] All other parameters were maintained constant, and at their nominal value. Using the MDNBR data from each steady-state simulation, sensitivity was computed with the units of 'percent-per-percent'. (i.e. percent change in MDNBR divided by percent change in parameter)

---

[s] The definition of the Engineering Enthalpy Rise Hot Channel Factor is given as:

$$F_{\Delta H}^{E} = \frac{\Delta Enthalpy_{hot-channel}}{\Delta Enthalpy_{core-average}}$$

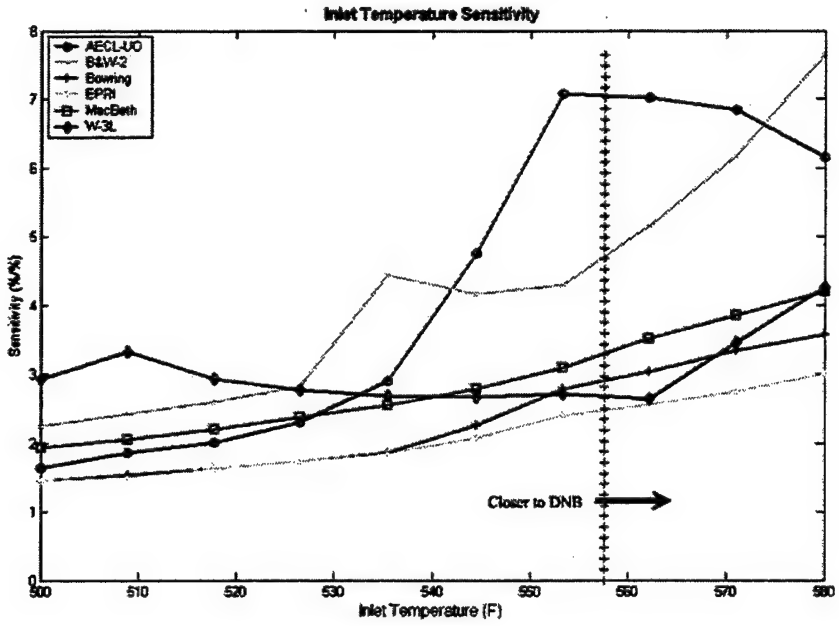
[t] A total of 100 VIPRE runs is therefore made for each of the five parameters (500 runs), for each CHF correlation (6 correlations for a total of 3000 VIPRE runs). The Matlab-VIPRE Interface was used for this analysis that would otherwise have taken several days of intensive effort if performed 'by hand'. Rather, these analysis were scripted and run during a single evening on a personal computer that was otherwise unused during this time.

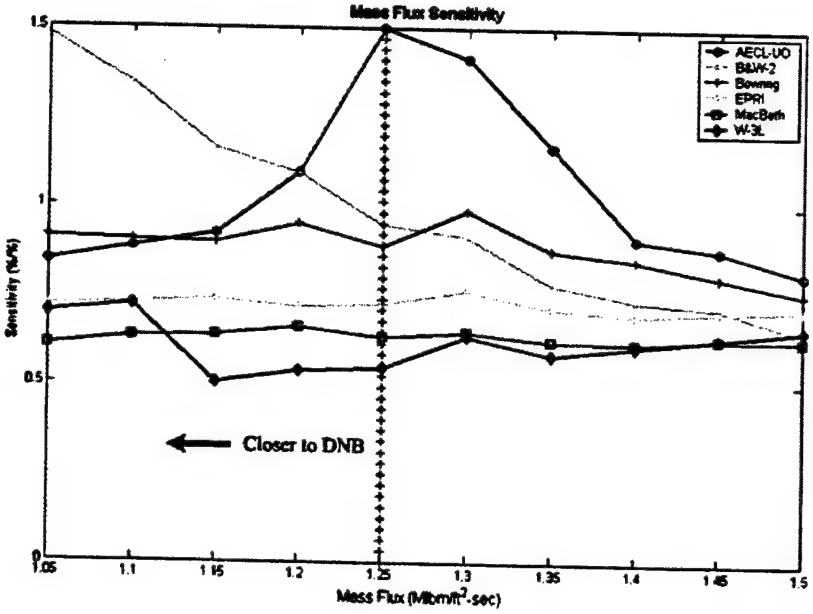
Parameter	Minimum Value	Nominal Value	Maximum Value
Inlet Temperature	500 °F	557.6 °F	580 °F
Mass Flux	$1.05\ \frac{MLbm}{ft^2 - hr}$	$1.253\ \frac{MLbm}{ft^2 - hr}$	$1.5\ \frac{MLbm}{ft^2 - hr}$
System Pressure	2150 psia	2259.4 psia	2375 psia
Linear Power	$2.6\ \frac{KW}{ft}$	$3.04\ \frac{KW}{ft}$	$3.5\ \frac{KW}{ft}$
$F_{\Delta H}^{E}$	1.0	1.02	1.04

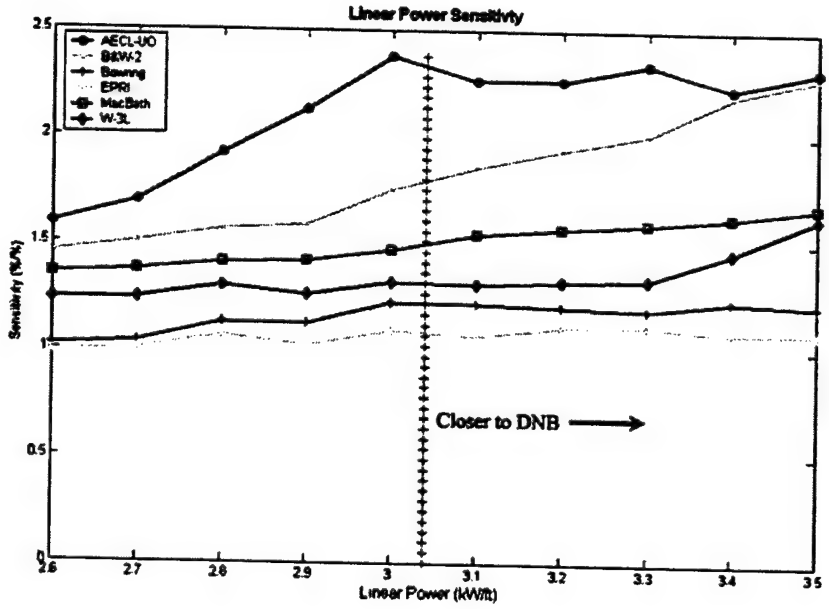
Table 9: Range of analysis for sensitivity study.

The sensitivity plots for this procedure are provided over the next several figures.[u] The nominal value of each parameter is marked with a cross-hatched line on each plot.

[u] An example scrip‘ used to generate these uncertainties is provided in the appendices.


Inlet Temperature Sensitivity
AECL-UO
B&W-2
Bowring
EPRI
MacBeth
W-3L
Sensitivity (%/%)
Inlet Temperature (F)
Closer to DNB
0
1
2
3
4
5
6
7
8
500
510
520
530
540
550
560
570
580


Mass Flux Sensitivity
AECL-UO
B&W-2
Bowng
EPRI
MacBeth
W-3L
Closer to DNB
Sensitivity (%/%)
Mass Flux (Mlbm/ft²-sec)


Linear Power Sensitivity
AECL-UO
B&W-2
Bowng
EPRI
MacBeth
W-3L
Closer to DNB
Sensitivity (%/%)
Linear Power (kW/ft)

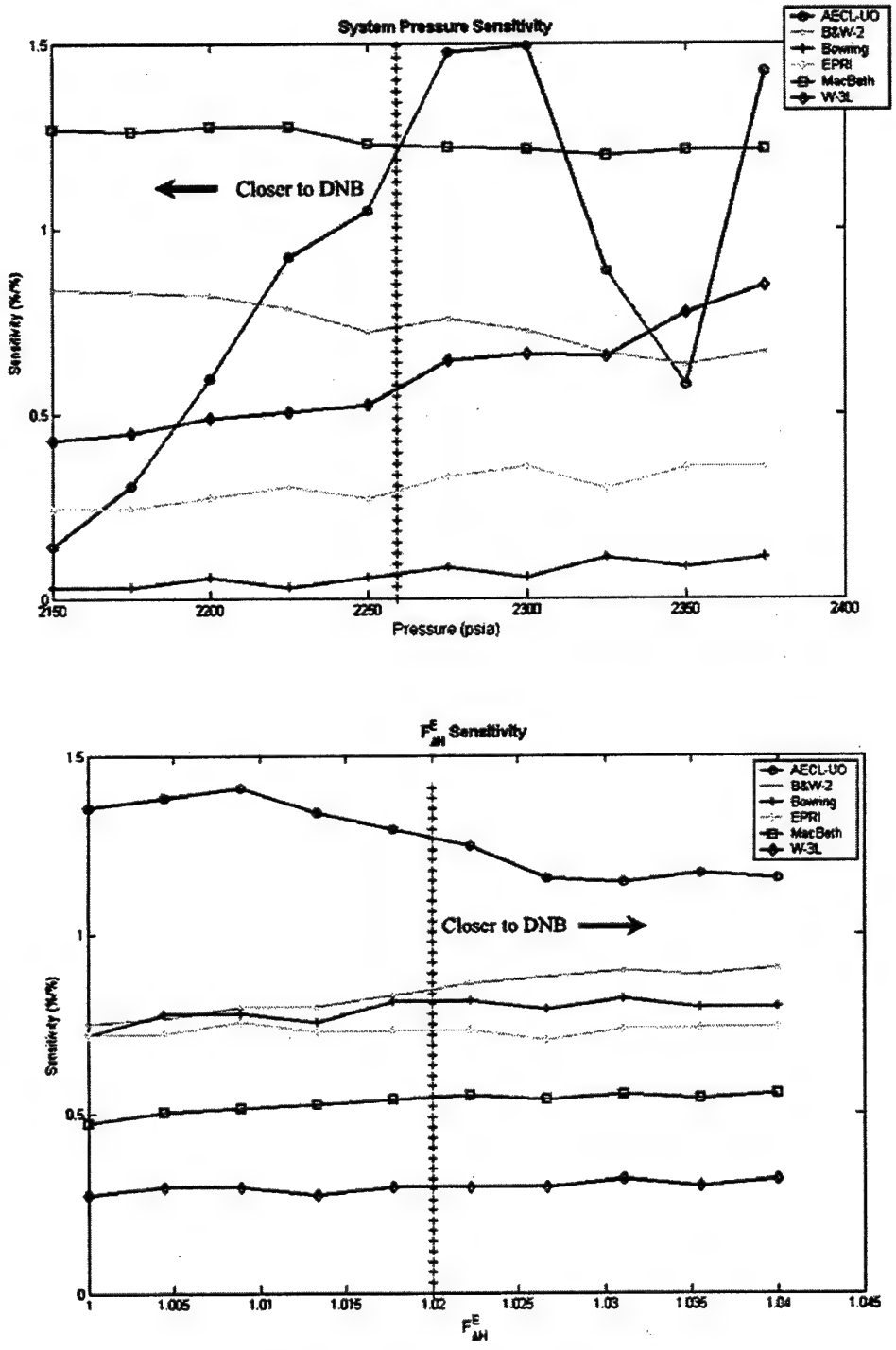


Figure 20: Parameter Sensitivities

It is notable that the sensitivity of most of the correlations is increased as the parameter is moved 'closer' to the DNBR limit – for example, as inlet temperature is

increased, the temperature sensitivity in increased. This has been found to be a feature common among the correlations studied in this sensitivity analysis. Also interesting to note is the difference in sensitivity between different CHF correlations. For example, the sensitivity of B&W-2 to all parameters is 'stronger' than for W-3L. This observation can be linked to the fact that, in Figure 10, the B&W-2 correlation has a steeper slope than the W-3L correlation. This indicates that the B&W-2 has, in general higher power sensitivity than the W-3L correlation. Without calculation one would guess that B&W-2 then also has higher power sensitivity than all of the other correlations considered. Numerical results will show that this is in-fact the case.

The above observation led to the second sensitivity analysis methodology that was employed for this study. The problem lies in the fact that, as will be shown in the Uncertainty Analysis section, for the ITDP, a single sensitivity factor must be selected, though it is clear from the previous figures that none of the sensitivities are constant. If a sensitivity factor is chosen that is too small, then the analysis is non-conservative.[v] If a sensitivity factor is chosen that is too large, then the analysis is over-conservative, resulting in a loss of power margin. Clearly, appropriate choice of the sensitivity factors is an important step in the ITDP that will be subject to close scrutiny from both regulators and potential customers. It may seem to be over-conservative to simply choose the largest sensitivity factor within the range considered for use in the ITDP since the range considered is quite broad compared to the range of parameters anticipated during plant operation. It is quite unlikely that any of the parameters considered would be so far from their nominal value. On the other hand, it would seem arbitrary to pick any value other than the highest achieved sensitivity – who decides what value should be chosen?

The question to ask is: If the sensitivity is increased, as the plant is closer to DNB, then couldn't the inlet temperature sensitivity be increased if the mass flux was reduced? (As opposed to increasing only inlet temperature – since reducing the mass flux brings the core even 'closer' to DNB and thus, one could expect the sensitivity of the correlation

[v] As will be shown, a lower sensitivity would then result in a lower variance for the DNBR – which would be a random variable for the ITDP analysis. The lower variance is non-conservative as compared to a higher variance.

to increase) To test for this behavior a second methodology was used. A tighter range was selected for each parameter. This range is given in Table 10.[w]

Parameter	Range of Interest
Inlet Temperature	± 4 °F
Mass Flux	± 5 %
Linear Power	± 2 %
System Pressure	± 30 psia
$F_{\Delta H}^{E}$	1.02 ± .02

Table 10: Parameter ranges for combinatorial sensitivity analysis procedure.

Three points were chosen in this range for each parameter. The sensitivity was found for each parameter for every combination of parameters.[x] The procedure was repeated for all CHF correlations. While this study requires an enormous number of VIPRE runs, the task was scripted[y] using the Matlab-VIPRE Interface described in chapter 2. Aside from the task of writing the script specifying the operations to be

[w] This is the same range that is used for the Monte Carlo uncertainty analysis procedure.

[x] There are 5 parameters, each that are sampled at three points within the range of interest. There are 3^5 permutations of parameter values. At each permutation, 3 VIPRE runs are performed to find the sensitivity. This process must be repeated for each of the 5 parameters, for each of the 6 CHF correlations considered for a total of: $3^5 \times 3 \times 5 \times 6 = 21{,}870$ VIPRE runs. To perform this by manually adjusting the VIPRE input file, executing VIPRE, and extracting the resulting data would require roughly 200 man-hours of work that would translate into a nearly insurmountable obstacle for the lone graduate student. By contrast, using the scripted interface, this calculation could be performed in roughly one day with the computer.

[y] Example script included in the appendices.

performed, the analysis was completed automatically. The results are presented in the form of a histogram for each parameter studied.

Since the data generated from this analysis is rightly placed in five dimensions (since there are 5 uncertain parameters), it was impossible to present the data in a surface or volume plot. A histogram was chosen to display this data in order to give the user a better feel for the behavior of a parameter's sensitivity around its nominal value.[z] In most cases, as shown by the histograms, the sensitivity is very close to that which would be achieved with all parameters at their nominal values. (The correlation sensitivity taken when all parameters are at their nominal value is shown as a cross-hatched line in each plot.) On the other hand, there are some combinations of plant conditions that leads to a sensitivity that is significantly higher. It is interesting that sensitivities can nearly double in magnitude even though plant conditions are all still close to nominal value.

[z] Thought this method of presentation is disadvantaged in that it does not show what position within the parameter space result in particular values of sensitivity.

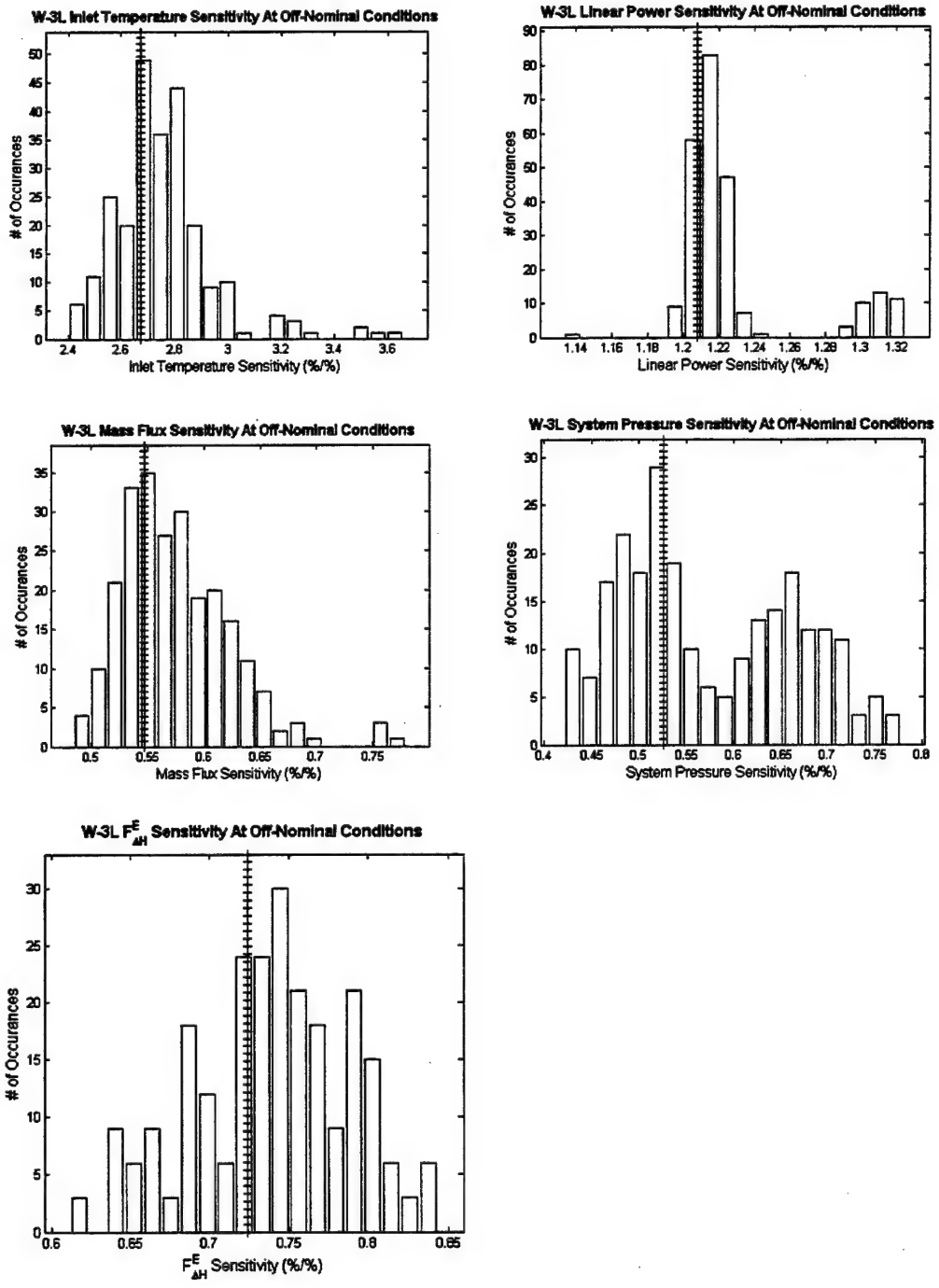


Figure 21: W-3L Correlation sensitivities.

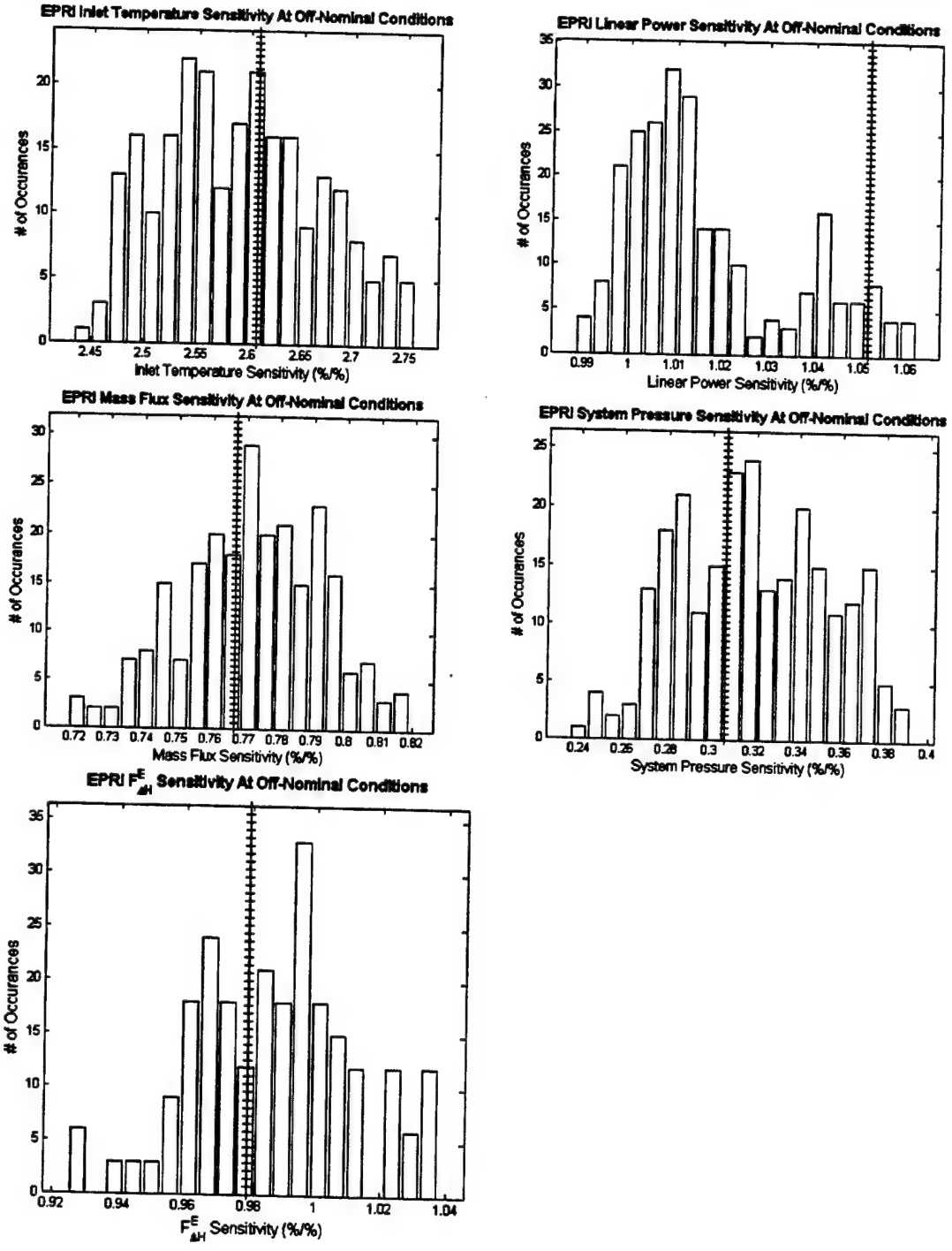


Figure 22: EPRI Correlation Sensitivities

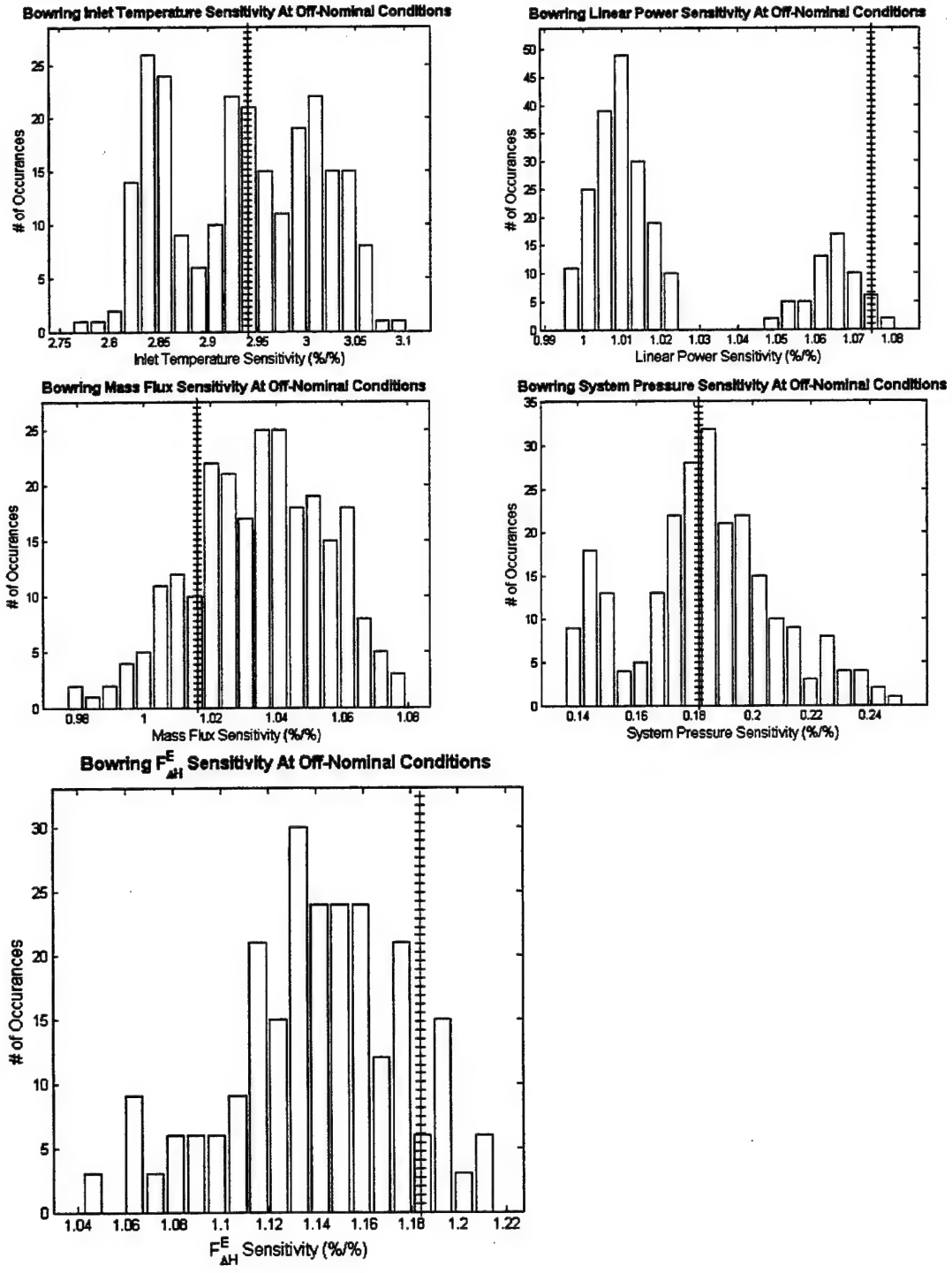


Figure 23: Bowring Correlation sensitivities.

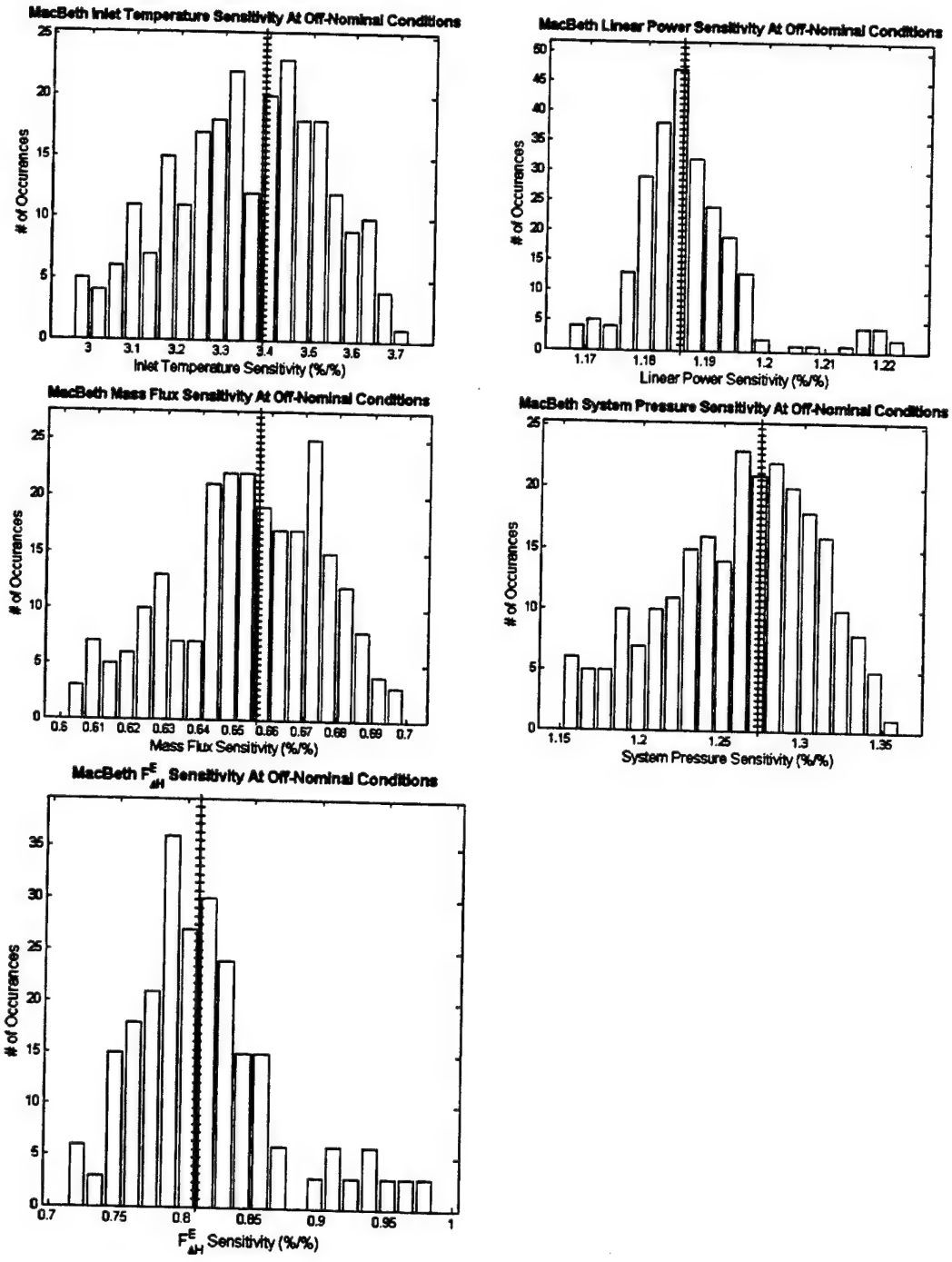


Figure 24: MacBeth Correlation sensitivities.

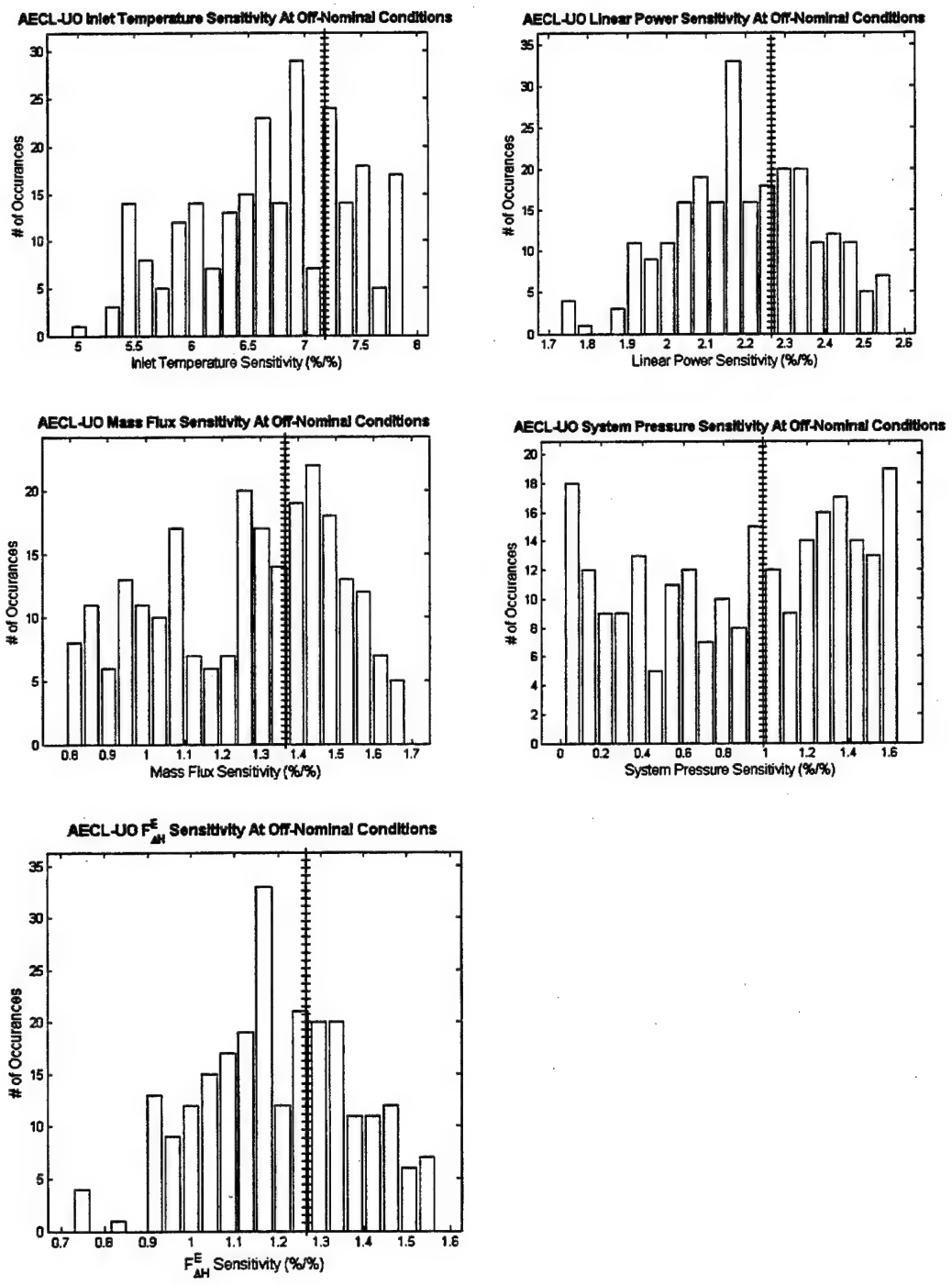


Figure 25: AECL-UO Look-Up Table Correlation Sensitivities.

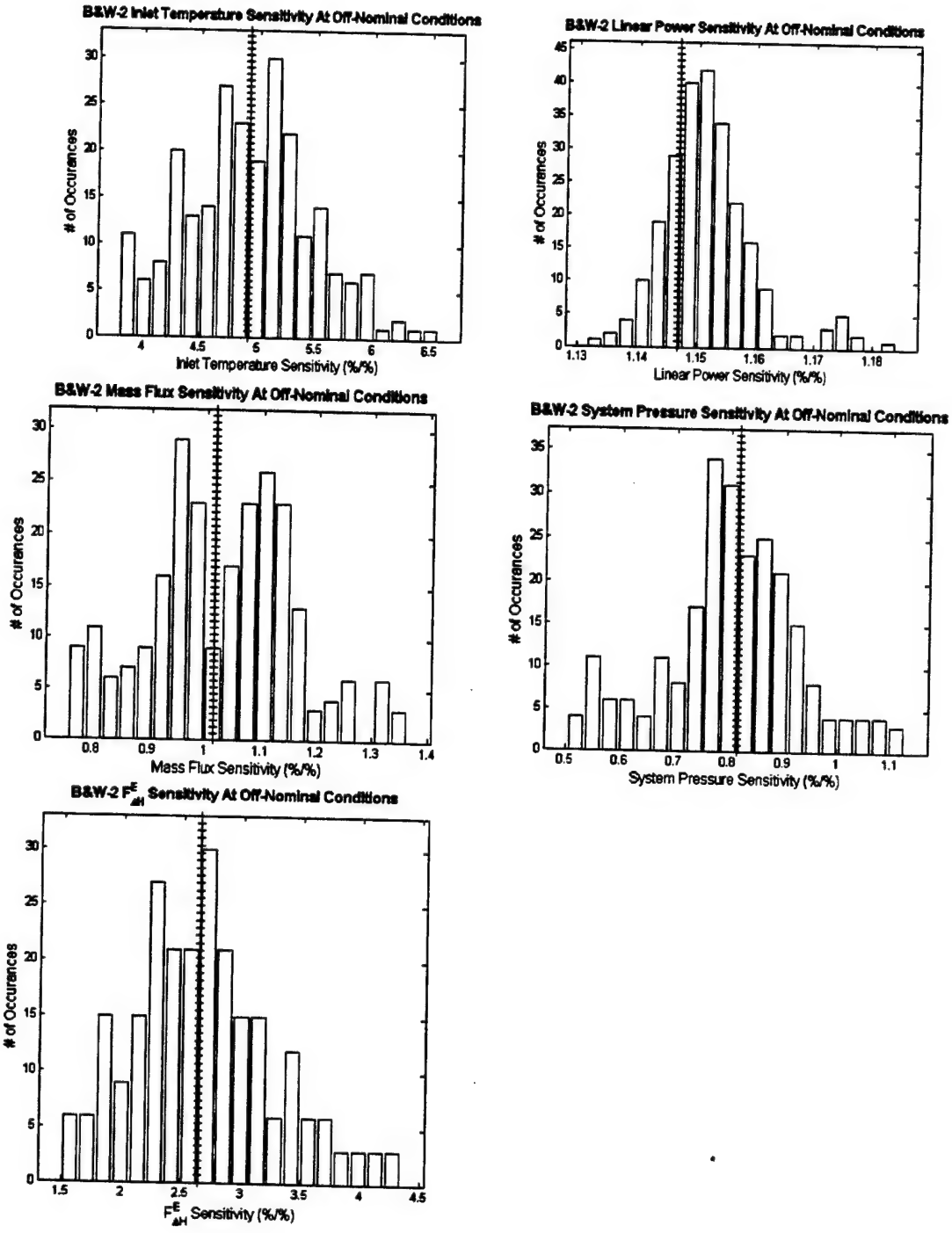


Figure 26: B&W-2 Correlation Sensitivities.

The numerical values of the maximum sensitivity found using this procedure are summarized in Table 11.

	Sensitivity, s_i $\%\Delta DNBR / \%\Delta parameter$					
Parameter	**EPRI**	**Bowring**	**B&W-2**	**W-3L**	**MacBeth**	**AECL-UO**
Mass Flux	0.81906	1.0794	1.481	0.778	0.69928	1.6833
Linear Power	1.0623	1.0808	2.32	1.601	1.2233	2.5673
Inlet Temperature	2.7596	3.1025	7.5	3.65	3.7275	7.8802
System Pressure	0.39185	0.25114	0.84	0.777	1.361	1.6405
$F_{\Delta H}^{E}$	1.0377	1.2148	1.8151	0.595	0.98569	1.5670

Table 11: Maximum Sensitivity.

As can be seen, the parameter sensitivities vary significantly from parameter to parameter and from correlation to correlation. Analysis of the inputs provided in generating the above histograms reveals that in every case, the maximum sensitivity was found with plant conditions that place the core closest to CHF – low: mass flux and system pressure, high: linear power, inlet temperature, and engineering enthalpy rise hot channel factor.

3.8. Uncertainty Analysis

Uncertainty is present in most every engineering design. For nuclear reactor designs, this uncertainty must be dealt with in such a way as to provide 'acceptable' confidence that the core will, in reality, operate in a manner that is farther from thermal limits than that which is predicted from numerical computations. The meaning of 'acceptable' has changed over the decades. In the nearly 50 years of commercial reactor design experience, there has been a steady evolution of methodologies for conservatively handling uncertainty. The methodologies listed in this section are not intended to be exhaustive, nor do I claim that they are representative of every scheme that has ever been formulated. This sampling does however touch over the extreme ends of the continuum and I think captures the upper and lower bounds of conservatism. A general overview of these design practices is provided in chapter 8 of reference [23].

Standard Thermal Design Procedure

The Standard Thermal Design Procedure (STDP) is one of the earliest techniques employed to handle uncertainty. This methodology is sometimes referred to as the *Cumulative* or *Product Cumulative* approach. Each design parameter subject to uncertainty is assumed to be simultaneously at its worst possible value from a DNB perspective. In this way the analyst can 'guarantee' that the actual in-core conditions are no worse than those postulated in the STDP methodology, and hence, the results obtained are, within the analytical assumptions made, conservative. The parameters under consideration and their worst-case values have already been presented in Table 9.[aa] For the STDP, the transient is simulated under these conditions, and the MDNBR obtained is compared to the DNBR limit for the given DNB correlation.

Locked Rotor/Shaft Shear

The results of the analysis of the LR/SS transient using the STDP are presented here:

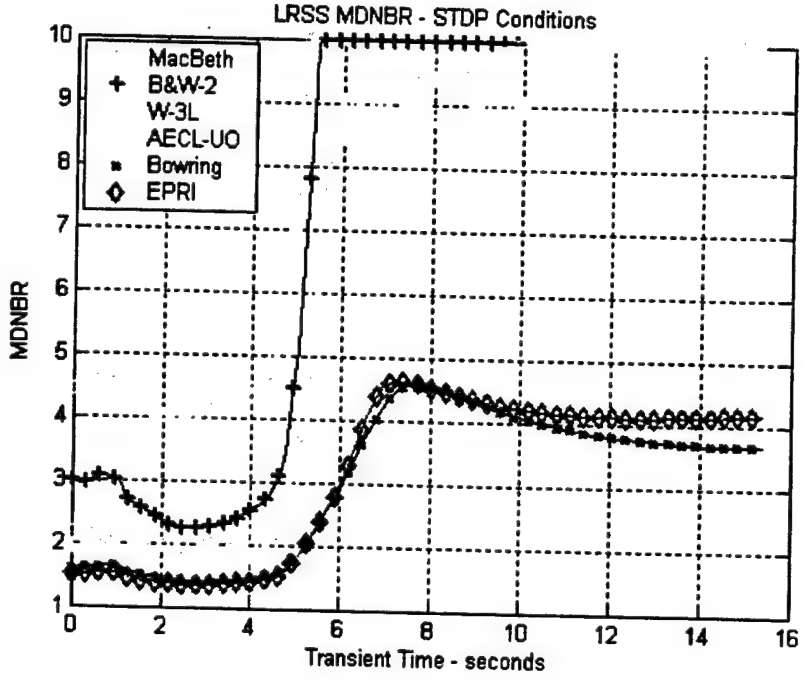


Figure 27: 17 x 17 Core, LR/SS Standard Thermal Design Procedure

[aa] Note that only process parameter uncertainties and engineering uncertainties are included as an illustration of the methodology.

Partial Loss of Flow – 4 of 8

The results of the PLOFA transient using the STDP are presented here:

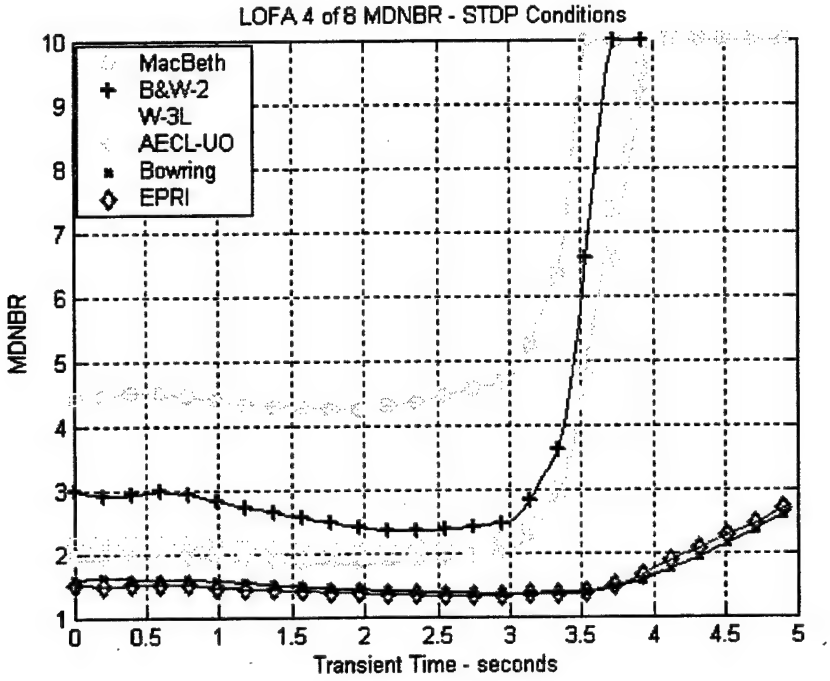


Figure 28: 17x17 PLOFA 4 of 8 STDP.

Complete Loss of Flow

The results of the CLOFA transient using the STDP are presented here:

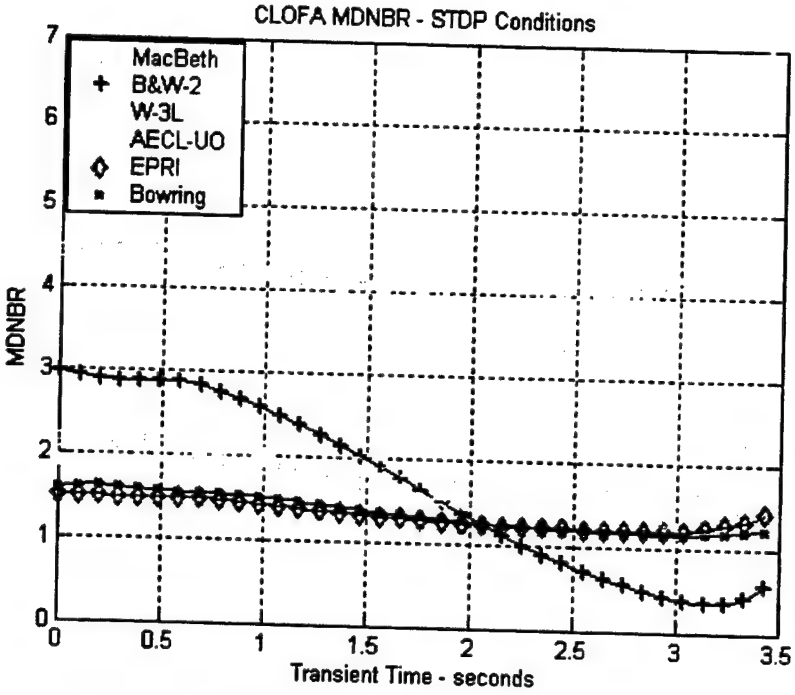


Figure 29: 17x17 CLOFA STDP.

A summary of MDNBR transient results using the STDP is provided in Table 12.

	MacBeth	B&W-2	W-3L	Bowring	EPRI	AECL-UO
LRSS	4.069	2.249	1.961	1.363	1.417	1.711
PLOFA	4.243	2.350	2.021	1.360	1.333	1.817
CLOFA	3.868	0.356	1.183	1.2	1.157	1.468

Table 12: Summary of STDP Transient Results.

Notice that every correlation except for MacBeth and AECL-UO Look-up Table result in a MDNBR of less than 1.30 for the CLOFA transient using the STDP.

Improved Thermal Design Procedure

This procedure is discussed in detail in reference [22]. The assumption fundamental to this methodology is that the uncertainties for all design parameters are mutually independent. Instead of assuming all parameters are at their worst-case value simultaneously, it is assumed that each parameter is an independent random variable. The DNBR, that is a function of these parameters, is therefore a random variable itself, with statistical properties related to the uncertain parameters. The goal of this procedure is to define a new MDNBR Limit that takes into account the parameter variability, and

the sensitivity of the correlation MDNBR to variations in each parameter and provides 95/95 confidence that DNB is not reached when the MDNBR for the transient is above this limit.

A parameter 'y' is defined to be the DNBR uncertainty factor in the following manner:

Equation 11

$$y = \frac{DNBR(\text{variable})}{DNBR(\text{nominal})}$$

The denominator of Equation 11 refers to the DNBR calculated, for the DNB correlation of interest, for the core as modeled with all uncertain parameters at their nominal, best-guess values. The numerator refers to the value of DNBR as affected by the randomly varying uncertain parameters. No assumptions are made regarding the exact distribution of these random variables, only that they posses a finite mean value (μ), and a variance (σ^2). This uncertainty factor, y, can be formally related to changes in design parameters, $\{x_1,\ldots,x_n\}$, and sensitivities, $\{s_1,\ldots,s_n\}$, as follows:

Equation 12

$$\frac{dy}{y} = s_1\frac{dx_1}{x_1} + s_2\frac{dx_2}{x_2} + \ldots + s_n\frac{dx_n}{x_n}$$

The sensitivity terms are defined to be the percent change in DNBR per percent change in the uncertain variable n. The design parameters $\{x_1,\ldots,x_n\}$ are defined to be those core design parameters that are subject to uncertainty, and have an impact on the core DNBR. If all but the i'th uncertain parameter is taken to be constant, the Equation 2 can be simplified as below where each x_i represents a design parameter subject to uncertainty:

Equation 13

$$\frac{dy}{y} = s_i\frac{dx_i}{x_i} \quad \text{or} \quad s_i = \frac{x_i}{y}\frac{dy}{dx_i}$$

If all of the uncertain parameters were at their nominal value, by inspection of Equation 11, one can see that the value of y is unity. What we are interested in is the behavior of y when all of the uncertain parameters are at some perturbation from their nominal condition. To analyze this behavior, we will start by formally expanding y in a Taylor series about its mean value[bb]:

Equation 14

$$y = \mu_y + \frac{\partial y}{\partial x_1}(x_1 - \mu_1) + \frac{\partial y}{\partial x_2}(x_2 - \mu_2) + \ldots + \frac{\partial y}{\partial x_n}(x_n - \mu_n) + \text{higher order terms}$$

The required result for the ITDP is an expression that gives μ_y and σ_y^2, as a function of the mean and variance of the uncertain parameters $\{x_1, \ldots, x_n\}$. Before this calculation is made, some manipulations and simplifying assumptions are required.

First, multiply and divide both sides of Equation 4 by μ_y. Multiply and divide each term on the right-hand-side of Equation 4 (with the exception of the μ_y) by the respective mean value of each random variable. We make the standard assumption that, for this expansion, the deviations from the mean value will be small, and therefore the higher order terms will be of small magnitude and can therefore be ignored. The result is:

Equation 15

$$\left(\frac{\mu_y}{\mu_y}\right) y = \left(\frac{\mu_y}{\mu_y}\right)\mu_y + \mu_y \frac{\partial y}{\partial x_1}\frac{\mu_1}{\mu_y}\left(\frac{x_1 - \mu_1}{\mu_1}\right) + \mu_y \frac{\partial y}{\partial x_2}\frac{\mu_2}{\mu_y}\left(\frac{x_2 - \mu_2}{\mu_2}\right) + \ldots + \mu_y \frac{\partial y}{\partial x_n}\frac{\mu_n}{\mu_y}\left(\frac{x_n - \mu_n}{\mu_n}\right)$$

$$y = \mu_y + \mu_y \frac{\partial y}{\partial x_1}\frac{\mu_1}{\mu_y}\left(\frac{x_1 - \mu_1}{\mu_1}\right) + \mu_y \frac{\partial y}{\partial x_2}\frac{\mu_2}{\mu_y}\left(\frac{x_2 - \mu_2}{\mu_2}\right) + \ldots + \mu_y \frac{\partial y}{\partial x_n}\frac{\mu_n}{\mu_y}\left(\frac{x_n - \mu_n}{\mu_n}\right)$$

[bb] We will assume that $y(x_1, \ldots, x_n)$ is a continuous function of the uncertain design variables with continuous derivatives as required for Taylor Series expansions.

Notice now that, on the right hand side of Equation 15, with the exception of the first term, each other term has the $\frac{\partial y}{\partial x_i}\frac{\mu_i}{\mu_y}$ component. This mathematically describes the sensitivity that was introduced in Equation 13, but evaluated with both y and the i'th uncertain parameter, x_i at its nominal (mean) value. Unfortunately, we have no way to analytically obtain the partial derivative of y with respect to the uncertain parameters x_i. While it is true that it is possible to uniquely map a given set of uncertain parameters $\{x_1 \ldots x_n\}$ to a specific value of y, this mapping can only be obtained with the use of a detailed plant model and thermal-hydraulic analysis code (VIPRE) – there is no way to simply write down the function y, much less find it's partial derivatives. As an approximation, this quantity will be replaced by the numerically computed sensitivity,[cc] Strictly speaking this is, of course, wrong,[dd] but this converts Equation 15 into a linear algebraic function of the random variables $\{x_1,\ldots,x_n\}$. The result now is:

Equation 16

$$y = \mu_y + \mu_y s_1(\frac{x_1-\mu_1}{\mu_1}) + \mu_y s_2(\frac{x_2-\mu_2}{\mu_2}) + \ldots + \mu_y s_n(\frac{x_n-\mu_n}{\mu_n})$$

For this equation, since $\{\mu_1,\ldots,\mu_n\}$ and $\{s_1,\ldots,s_n\}$ are assumed to be known and constant and μ_y is by construction unity, all of the terms on the right hand side of Equation 16 are known. The desired quantity, namely, σ_y^2, can now be obtained directly as Equation 17 based on standard results pertaining to the variance of linear functions of random variables[24],[25]:

Equation 17

$$\sigma_y^2 = \mu_y^2 s_1^2\left(\frac{\sigma_1}{\mu_1}\right)^2 + \mu_y^2 s_2^2\left(\frac{\sigma_2}{\mu_2}\right)^2 + \cdots + \mu_y^2 s_n^2\left(\frac{\sigma_n}{\mu_n}\right)^2$$

[cc] The details of the computation of the sensitivities are provided in the preceding section "Sensitivity Analysis". An example of the computer code used for this purpose is provided in the appendices.

[dd] In other words, the sensitivity is not constant – it is not independent of the value of it's associated uncertain parameter, nor is it independent of the *other* uncertain parameters as was demonstrated in the previously described sensitivity analysis.

Now dividing through by μ_y^2, we obtain:

Equation 18

$$\left(\frac{\sigma_y}{\mu_y}\right)^2 = s_1^2\left(\frac{\sigma_1}{\mu_1}\right)^2 + s_2^2\left(\frac{\sigma_2}{\mu_2}\right)^2 + \ldots + s_n^2\left(\frac{\sigma_n}{\mu_n}\right)^2$$

This is the main result of the ITDP. Each s_i is the sensitivity for the i'th parameter for the given DNB correlation, and σ_i/μ_i is the coefficient of variation (COV) for the design parameter.

To employ this procedure, all that is needed are the parameters called for in the right hand side of Equation 18. Since μ_y is, by construction, unity, Equation 18 can be used to solve for σ_y^2 under the combined influence of each random design parameter.

Since we have not made any reference to the actual underlying distribution of each of the uncertain design parameters, it is a legitimate argument to question the underlying distribution of the parameter y. There is a well-established, commonly studied, body of mathematical results in the case that all of the uncertain parameters are normally distributed. In the more general case, where the uncertain design parameters are of arbitrary distribution, we will assume, based on the Central Limit Theorem,[24] that y is normally distributed.

Using this variance, and the assumption that y is normally distributed, the upper 95 % confidence μ_y is determined by reducing μ_y 1.645 standard deviations. Using the notation of reference [22], this gives a factor $0 < F_u < 1$.[ee]

Equation 19

$$F_u = \mu_y - 1.645 \cdot \sigma_y = 1 - 1.645 \cdot \sigma_y$$

[ee] Strictly speaking, equation 9 only ensures that $F_u < 1$ since σ_y is non-negative for any probability distribution. It is not considered plausible that parameter sensitivities would be such that $\sigma_y > 1.65$ so that F_u would be negative.

The correlation DNBR is then divided by this factor, resulting in the ITDP limit DNBR.

Equation 20

$$DNBR_{ITDP} = \frac{DNBR_{correlation-limit}}{F_u}$$

The ITDP was employed in the following analysis of each casualty event. The W-3L, B&W-2, EPRI, AECL-UO and MacBeth correlations were used. The preliminary inputs necessary for this analysis are:

- DNBR sensitivities for each correlation $\{s_1 \cdots s_n\}$;
- The MDNBR for each transient for design parameters at their nominal value (μ_y); and
- The postulated probability density function for the design parameters of interest and their associated statistical properties.

For illustration each design parameter was assumed to have a rectangular distribution (uniform probability) about it's nominal value. For a rectangular distribution, the variance is given by[25]:

Equation 21

$$\sigma^2 = \frac{(b-a)^2}{12}$$

Here, *b* and *a* are the distribution upper and lower bounds respectively. The numerical values of the statistical properties of these distributions are provided in Table 13.

Parameter	Mean Value	Range of Interest[ff]	σ^2	COV
Inlet Temperature	557.6 °F	± 4 °F	5.33	0.00414
Mass Flux	1.235 $Mlbm/ft^2-hr$	± 5 %	0.00127	0.0289
Linear Power	3.04 kW/ft	± 2 %	0.00123	0.0115
System Pressure	2259.4 psia	± 30 psia	300	0.0077
$F^E_{\Delta H}$	1.02	± .02	0.000133	0.0113

Table 13: Coefficient of Variation for Various Parameters.

For this analysis, the sensitivity factors obtained from the combinatorial sensitivity analysis method described in the previous section were used. These values are provided in Table 11. The resulting ITDP DNBR limits for the correlations used are presented in Table 14.

Correlation	F_u (Equation 19)	$DNBR_{ITDP}$ (Equation 20)
EPRI	0.941	1.382
Bowring	0.926	1.404
B&W-2	0.883	1.495
W-3L	0.944	1.377
MacBeth	0.940	1.382
AECL-UO Look-up Table	0.959	1.355

Table 14: ITDP DNBR Limits.

Putting this analysis into the same context as illustrated in Figure 8, an example is made of the W-3L correlation as shown in Figure 30. Here, the Design Condition and Safety Analysis Limit are yet to be determined, but the Correlation and Design Limit has been computed using Equation 19 and Equation 20 account for both correlation uncertainties, process parameters and engineering uncertainties.

[ff] Based on recommendations given in reference [22].

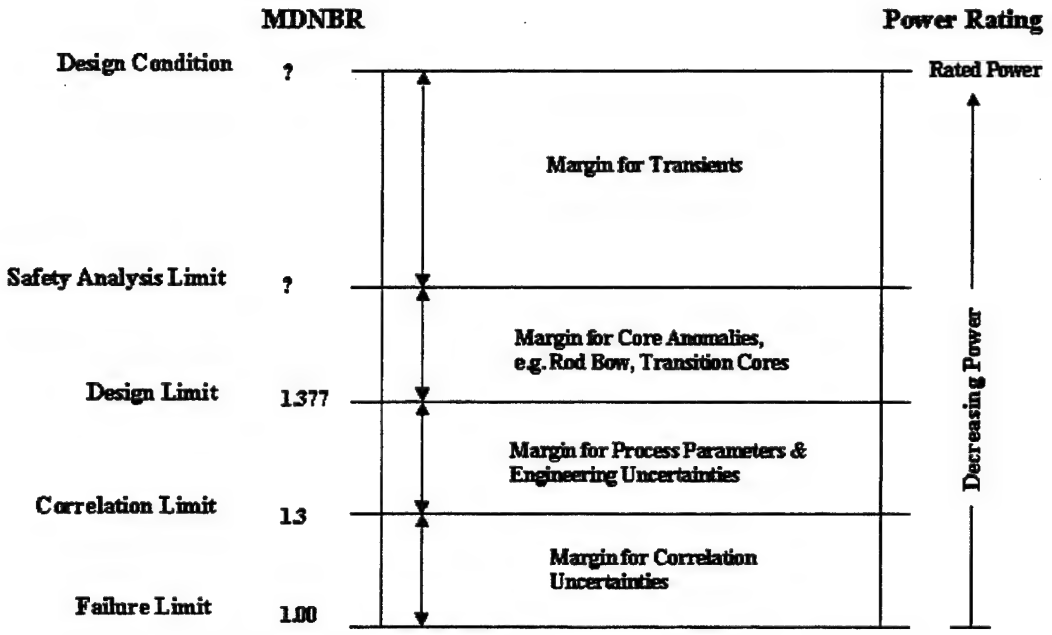


Figure 30: Illustration of Effect of Uncertainties for the W-3L Correlation.

It could be somewhat misleading to compare the ITDP DNBR limits from two different correlations. A higher ITDP DNBR limit is reflective of the higher sensitivity of a given correlation.[gg] However, for correlations of equal predictive capability, it would be, for the purposes of the ITDP, advantageous to have a correlation that exhibits the lowest possible sensitivity, therefore resulting in a lower DNBR limit.[hh]

A second note that should be made regarding the ITDP is the probability distribution used to describe the input parameter uncertainty. The ITDP uses only the mean and variance of the given distribution, but these two parameters do not represent all of the information contained within the statistical distribution. The rectangular distribution is parameterized by its computed parameters, but a normal distribution, for example, with the same parameters does not mathematically express the same meaning with respect to

---

[gg] I.e. the DNBR sensitivities $\{s_1 \cdots s_n\}$ tend to be larger in magnitude.

[hh] In practice this should not be a problem. T/H analysts in industry do not select from a stable of candidate correlations for a given fuel assembly. Instead, a proprietary in-house correlation that is finely tailored to the target fuel assembly is employed. The sensitivity characteristics of the correlation are beyond the control of the analyst. The point is only made that, all other things being equal, a very sensitive correlation – in particular, a correlation that has a widely varying (and large magnitude) sensitivity in many parameters will be penalized by the ITDP because of that sensitivity.

the parameter uncertainty. The ITDP however does not capture this difference. The significance of this difference will be demonstrated using the Monte Carlo Uncertainty Procedure where it will be shown that a different (less conservative) answer is obtained when using, as an example, a normal distribution with the same mean and variance as those of the rectangular distribution.

Monte Carlo Uncertainty Procedure

The Monte Carlo Uncertainty Procedure (MCUP) is conceptually much simpler than the ITDP. For this procedure, each probability distribution that describes the uncertain input parameters is sampled a large number of times – typically on the order of two to three thousand samples. These samples are then used as trial inputs to the VIPRE transient calculation.[ii] The resulting MDNBR calculations from all trial inputs are then analyzed for their statistical properties. We assume for this analysis that the resulting MDNBRs are normally distributed. This can be argued from the tenets of the Central Limit Theorem, on plausibility grounds by inspecting a histogram of the output data, or by performing a test of normality – all of which will be done here. The distribution parameters of the output are then adjusted to achieve the same 95/95 confidence interval obtained for the ITDP.

The core model used for this procedure is exactly the same as that used for the ITDP and STDP. This is in contrast to many of the other Monte Carlo-type procedures where a simplified core model is used.[26] Additionally, the output used to create the MDNBR statistics was generated directly from VIPRE. Many other Monte Carlo-type procedures use various methods for developing a response surface. In those other methods, such a response surface is in turn used to estimate the MDNBR that VIPRE *would have* computed for the given input parameters.[27] The response surface they used is then carefully and thoroughly benchmarked against a full VIPRE simulation, but is an approximation nonetheless.

[ii] The transient boundary condition data produced from RELAP is provided in a normalized form. Therefore, irrespective of the magnitude of initial core power, mass flow rate, pressure, or temperature, the shapes of the individual parameters will be the same through the transient. This assumption is justified by the relatively small perturbations made to the input data. If, for example, inlet temperature were 50 percent lower before the transient, it would probably be a poor choice to assume that the overall time-temperature profile is the same as that for the nominal case.

As a result of using the methodology as described, the computational requirements are not inconsiderable. For the desktop PC used for the majority of numeric computations presented in this report, approximately 1 day was required for every 1400 transient simulations. Some consideration was put into how many simulations should be run for each transient and for each correlation. The benefits obtained from more transient simulation runs are, in no particular order:

- More confidence that the parameter space has been sampled completely
- Smaller confidence intervals for the MDNBR statistical parameters
- The resulting MDNBR distribution has the greater appearance of 'normality'

Here a detailed description of the procedure is given. A detailed sample calculation is provided in the appendices:

- A number, typically 2000, random values with specified statistical properties are generated for the five parameters that will be modeled in a Monte Carlo fashion.[jj]
- The transient is run for five seconds for each transient.[kk] The MDNBR is recorded for each simulation run.
- After all simulations have been run, the sample mean MDNBR and standard deviation are computed.

These values are computed using standard statistical methods:

Equation 22

$$\hat{\mu}_{MDNBR} = \frac{1}{N}\sum_{i=1}^{N} MDNBR_i \quad \hat{\sigma}_{MDNBR} = \frac{1}{(N-1)}\left[\sum_{i=1}^{N} MDNBR_i^2 - N\mu_{MDNBR}^2\right]^{1/2}$$

Where:

$\hat{\mu}_{MDNBR}$ is the sample mean MDNBR,
$\hat{\sigma}_{MDNBR}$ is the sample standard deviation of MDNBR, and
N is the number of samples

- Using the chi-square distribution, compute the 95% upper confidence limit on the sample standard deviation.

[jj] For this application, random sampling of the parameter distributions is employed. More sophisticated sampling strategies such as stratified sampling or a Latin Hypercube Sampling (or modified version thereof) could be employed in the future as a way to possible reduce the number of samples required.
[kk] Experience has shown that, for the transients in this analysis, the MDNBR always occurs at approximately 2.5-3.0 seconds, so 5 seconds was deemed sufficient to ensure that the MDNBR was observed.

These values are also computed using standard statistical methods:

Equation 23

$$\sigma_{MDNBR} = \hat{\sigma}_{MDNBR}\left[\frac{N-1}{\chi^2}\right]^{1/2}$$

Where:

σ_{MDNBR} is the estimated population standard deviation[ll] of MDNBR,
χ_2 is the Chi-squared statistical parameter.

The Chi-squared parameter is a number that is dependent upon the confidence level desired for σ_{MDNBR} and the number of samples taken.

Equation 24

$$\chi^2(c) = \frac{1}{2^{n/2}\Gamma\left(\frac{N}{2}\right)} c^{(N/2-1)} e^{-c/2}$$

Where:

c is the desired confidence level
Γ is the standard Gamma function

These values can be obtained from pre-computed mathematical tables or programmed inverse functions. For the analysis in this thesis, the later route was taken.[mm]

- Using the 95% upper confidence limit standard deviation, compute the 95% lower confidence limit mean MDNBR. That is, the value that is below the actual MDNBR mean with 95% probability.

This is computed using the Student *t* distribution as follows:

[ll] In this context, the estimated population standard deviation represents the standard deviation that the core would actually have, to a specified degree of confidence, in the presence of the uncertain parameters that we have assumed.

[mm] Because of the character of the Chi-squared function, for analysis with sample sizes of greater than 1000, this step is actually skipped. Microsoft Excel workbook functions were used to obtain values for the Chi-squared parameter. For sample sizes above 1000, a floating point exception is returned, resulting in an error. As no tabulated values for the Chi-squared function were found with very large sample sizes, it was assumed therefore that at large values of n, that the sample standard deviation is equal to the population standard deviation.

Equation 25

$$\mu_{MDNBR} = \hat{\mu}_{MDNBR} - \frac{t\sigma_{MDNBR}}{\sqrt{N}}$$

Where:

μ_{MDNBR} is the estimated population mean MDNBR,
t is the Student t parameter,
$\hat{\sigma}_{MDNBR}$ is the estimated population standard deviation of MDNBR

The Student t parameter is computed in a way analogous to the Chi-squared parameter.

- Now adjust the 95% confidence mean MDNBR downward by 1.645 standard deviations (the 95% upper confidence standard deviation) to find the 95/95 confidence limits on MDNBR.

Equation 26

$$\mu_{95/95-MDNBR} = \mu_{MDNBR} - 1.645(\sigma_{MDNBR})$$

Because of the behavior of the Chi-squared and Student t distributions, the methodology will result in a more precise calculation of MDNBR as the number of random samples is increased. Unfortunately, the precision of the estimate of MDNBR only improves in a way proportional to the square root of the number of samples taken.[nn] The question that was to be answered early in the development of this methodology is: how many samples should be taken before the benefits of additional samples is outweighed by the computational difficulty.

Figure 31 is given to convey a more quantitative feel for the nature of the aforementioned trade-off. The upper plot shows a normalized histogram[oo] with a series of curves superimposed. Each curve represents a normal distribution with the 95 percent lower confidence limit mean MDNBR with the 95 percent upper confidence standard deviation. As is expected, for an increased number of samples, the curve becomes more

[nn] Therefore, to double the precision, four times as many samples must be taken.
[oo] The vertical bars are scaled such that the cumulative area of all bars is equal to one.

'pointed' to reflect the improved confidence in the computed mean and variance. The lower curve shows the corresponding 95/95 confidence mean MDNBR. As this lower curve levels off, no additional benefit is accrued by performing additional random samples. As can be seen, performing more than about 2000 sample runs has very little additional benefit.

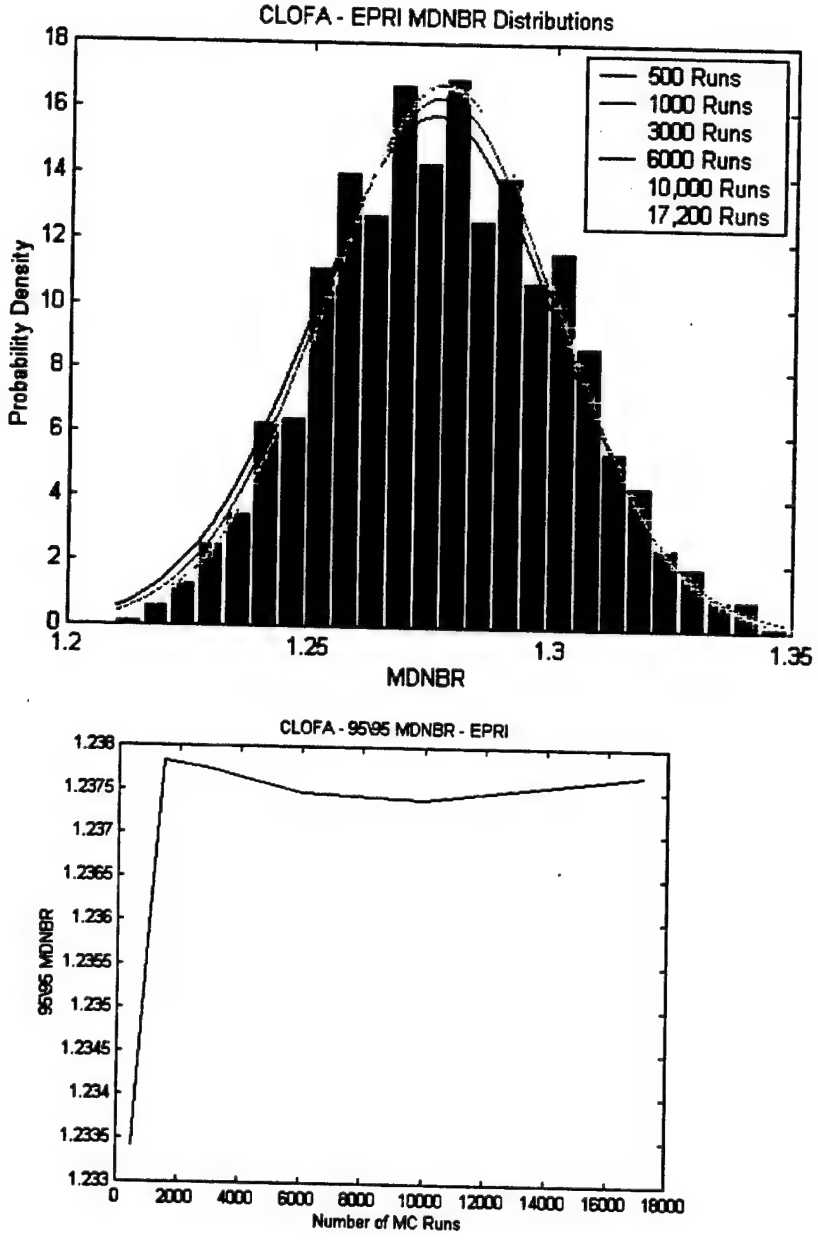


Figure 31: Results for large number of Monte Carlo samples for CLOFA using the EPRI correlation

MCUP Results

The following graphical results are given for the transient analysis using the MCUP. The graphical results are given in the form of a normalized histogram displaying the distribution of the sample data.[pp] Superimposed over the histogram is a cyan line showing the normal distribution with a mean value equal to the 95% lower confidence mean and 95% upper confidence standard deviation. A vertical magenta line is provided to highlight the position of the lower 5th percentile that represents the MDNBR limit for the specified correlation and transient.

LR/SS

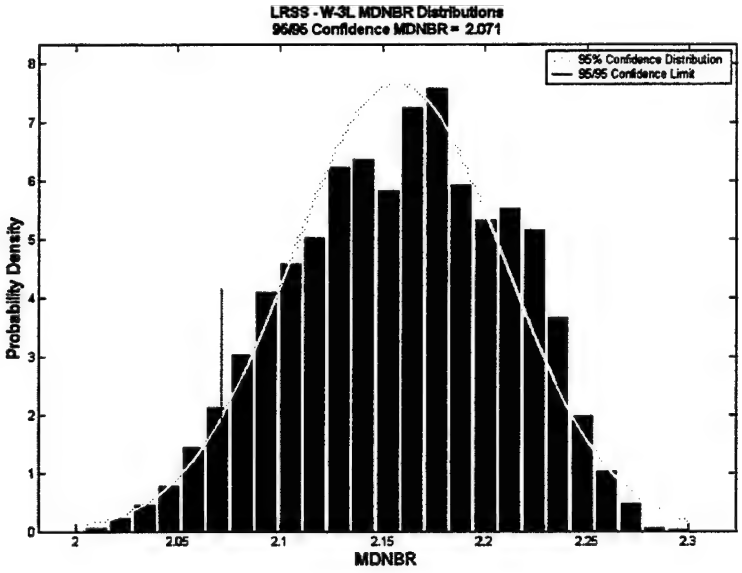


Figure 32: 17x17 W-3L LR/SS MCUP Results.

[pp] The 'sample data' is the collection of MDNBR results for the series of random inputs provided to VIPRE.

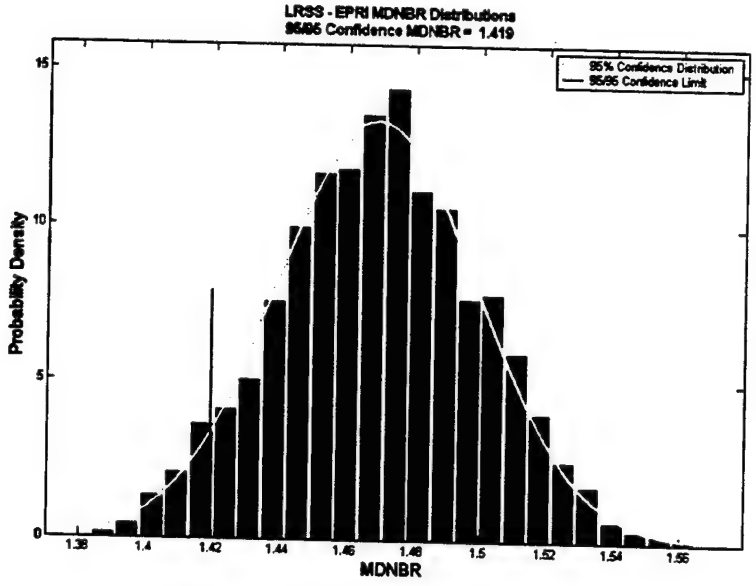


Figure 33: 17x17 EPRI LR/SS MCUP Results.

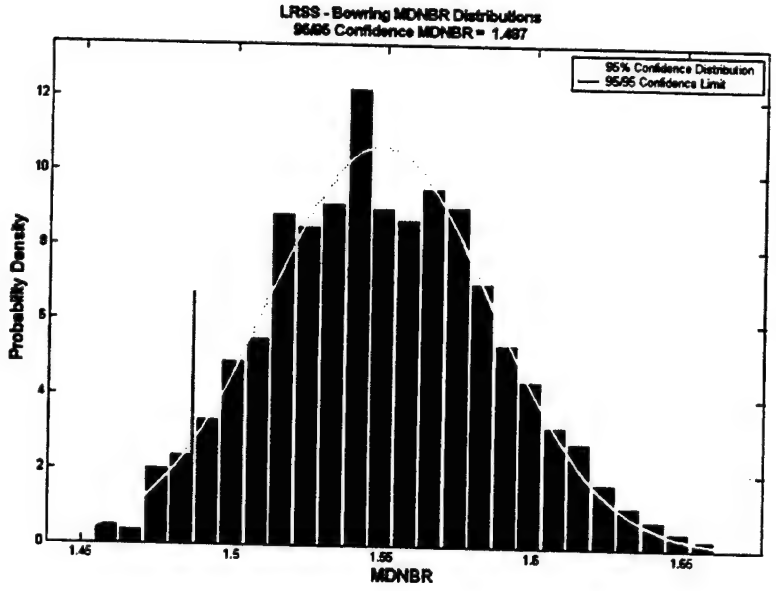


Figure 34: 17x17 Bowring LR/SS MCUP Results.

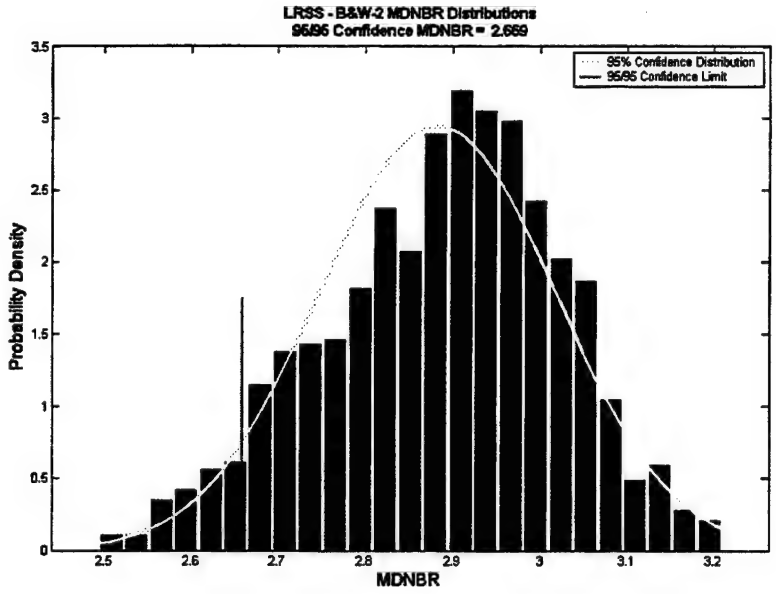


Figure 35: 17x17 LR/SS MCUP B&W-2 Results.

PLOFA 4 of 8 RCPs

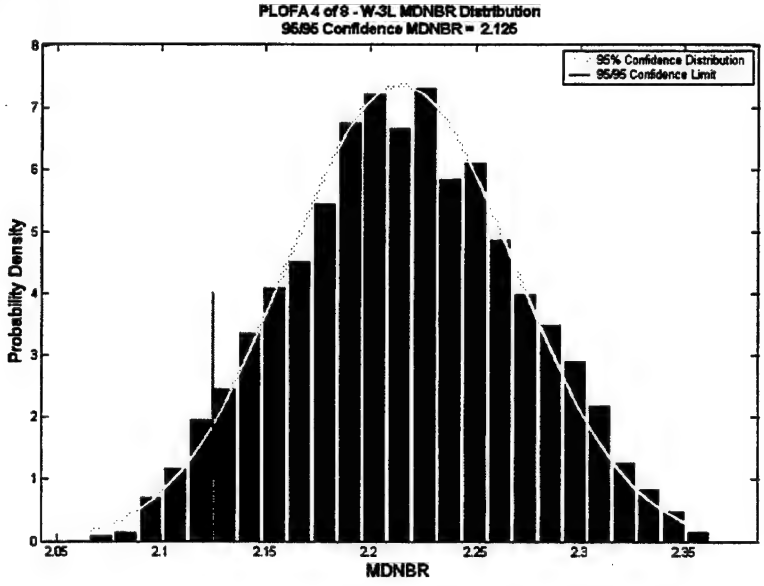


Figure 36: 17x17 PLOFA 4 of 8 RCPs MCUP W-3L Results.

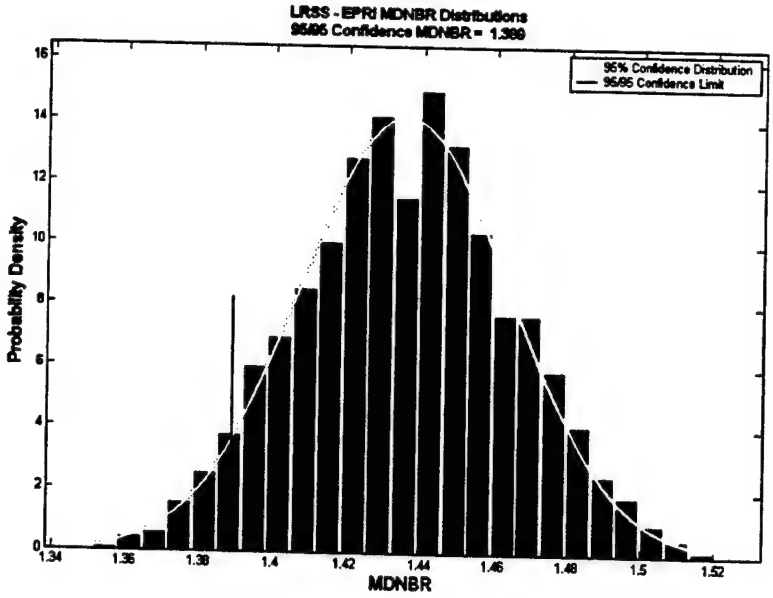


Figure 37: 17x17 PLOFA 4 of 8 RCPs EPRI Results.

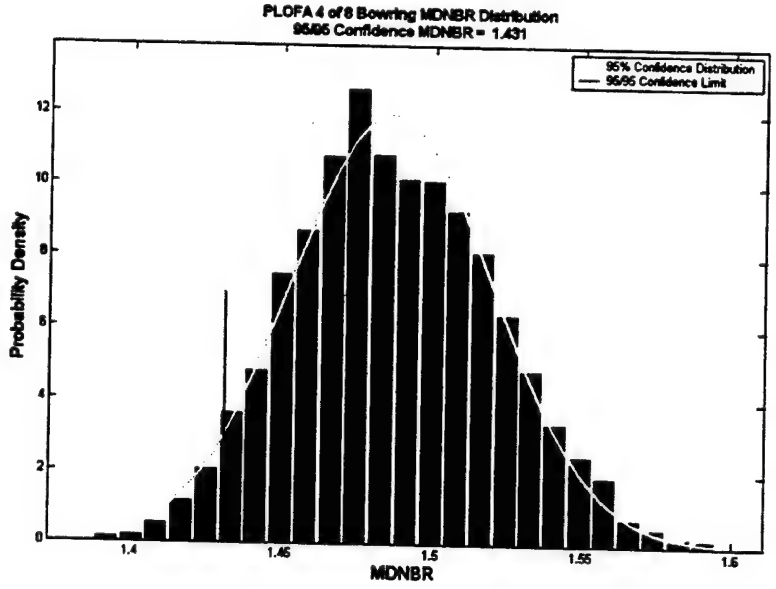


Figure 38: 17x17 PLOFA 4 of 8 RCPs Bowring Results.

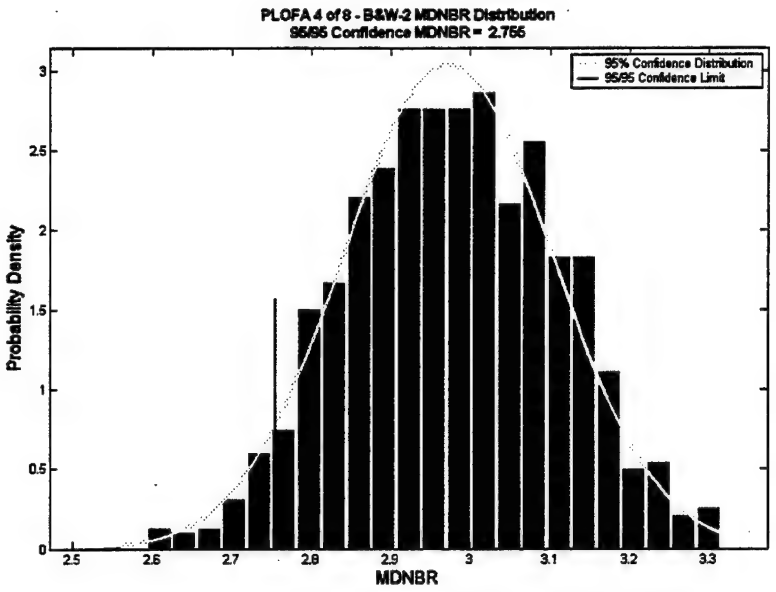


Figure 39: 17x17 PLOFA 4 of 8 RCPs B&W-2 Results.

CLOFA

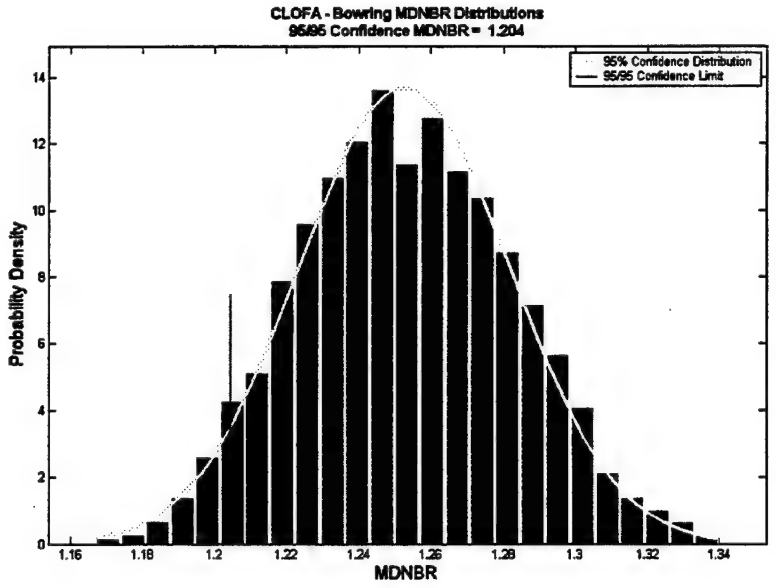


Figure 40: 17x17 CLOFA Bowring Results.

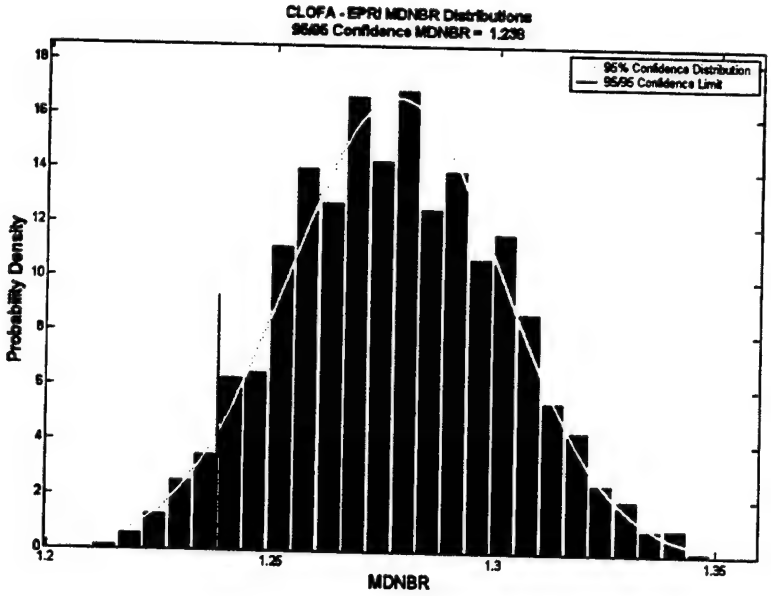


Figure 41: 17x17 CLOFA EPRI Results.

Chapter 4 Conclusions

4.1. Thermal Hydraulic Analysis Results

The analysis presented in this chapter yields two results: The first pertains to the T/H acceptability of the IRIS Open core. The second result is a comparison of the uncertainty analysis procedures utilized.

For the reference 1000 MWth IRIS Open core, a numerical summary of the transient T/H results is provided in the following tables.

Method	Correlation	MDNBR Limit	Computed MDNBR	Percent Above MDNBR Limit
STDP	EPRI	1.3	1.20	-7.692%
ITDP	EPRI	1.382	1.296	-6.223%
MCUP	EPRI	1.3	1.2360	-4.923%
STDP	Bowring	1.3	1.157	-11.000%
ITDP	Bowring	1.404	1.276	-9.117%
MCUP	Bowring	1.3	1.199	-7.769%
MC/Norm[a]	Bowring	1.3	1.202	-7.538%
STDP	W-3L	1.3	1.183	-9.000%
ITDP	W-3L	1.377	1.993	44.735%
STDP	B&W-2	1.32	0.356	-73.030%
ITDP	B&W-2	1.495	1.527	2.140%
STDP	MacBeth	1.3	3.868	197.538%
ITDP	MacBeth	1.382	4.308	211.722%
STDP	AECL-UO	1.3	1.468	12.923%
ITDP	AECL-UO	1.355	1.828	34.908%

Table 15: CLOFA Uncertainty Analysis Summary.

[a] MC/Norm refers to the use of MCUP with uncertain parameters being described by normal distributions with the same mean and variance as the same parameters described by a rectangular distribution.

Method	Correlation	MDNBR Limit	Computed MDNBR	Percent Above MDNBR Limit
STDP	EPRI	1.3	1.333	2.538%
ITDP	EPRI	1.382	1.459	5.572%
MCUP	EPRI	1.3	1.385	6.538%
MC/Norm[b]	EPRI	1.3	1.388	6.769%
STDP	Bowring	1.3	1.360	4.615%
ITDP	Bowring	1.404	1.515	7.906%
MCUP	Bowring	1.3	1.431	10.077%
STDP	W-3L	1.3	2.021	55.462%
ITDP	W-3L	1.377	2.238	62.527%
MCUP	W-3L	1.3	2.125	63.462%
STDP	B&W-2	1.32	2.350	78.030%
ITDP	B&W-2	1.495	3.055	104.348%
MCUP	B&W-2	1.32	2.755	108.712%
STDP	MacBeth	1.3	4.243	226.385%
ITDP	MacBeth	1.382	4.753	243.922%
STDP	AECL-UO	1.3	1.817	39.769%
ITDP	AECL-UO	1.355	1.848	36.384%

Table 16: PLOFA Uncertainty Analysis Summary.

[b] MC/Norm refers to the use of MCUP with uncertain parameters being described by normal distributions with the same mean and variance as the same parameters described by a rectangular distribution.

Method	Correlation	MDNBR Limit	Computed MDNBR	Percent Above MDNBR Limit
STDP	EPRI	1.3	1.363	4.846%
ITDP	EPRI	1.382	1.495	8.177%
MCUP	EPRI	1.3	1.419	9.154%
STDP	Bowring	1.3	1.417	9.000%
ITDP	Bowring	1.404	1.578	12.393%
MCUP	Bowring	1.3	1.487	14.385%
STDP	W-3L	1.3	1.961	50.846%
ITDP	W-3L	1.377	2.180	58.315%
MCUP	W-3L	1.3	2.071	59.308%
STDP	B&W-2	1.32	2.249	70.379%
ITDP	B&W-2	1.495	2.974	98.930%
MCUP	B&W-2	1.32	2.659	101.439%
STDP	MacBeth	1.3	4.069	213.000%
ITDP	MacBeth	1.382	4.588	231.983%
STDP	AECL-UO	1.3	1.711	31.615%
ITDP	AECL-UO	1.355	1.763	30.111%

Table 17: LR/SS Uncertainty Analysis Summary.

Because of restrictions on the use of proprietary data, this analysis has not incorporated all of the features that will be included in the final IRIS T/H design analysis. The following is a list of factors that would influence the results, but are not used:

- Only non-proprietary CHF correlations have been used. None of the correlations used in this analysis take into account all of the design features of the fuel assemblies that will be used in the core. In general, this is expected to impart a negative bias on the analysis.[c] The significant effort expended by fuel vendors to improve the thermal performance of their fuel has not been captured. This is likely to have the largest impact on the final T/H results. In any case, a proven correlation will be critical in the final T/H analysis required to obtain regulatory approval of the core design.
- Only non-proprietary coefficients for various heat-transfer and pressure-drop correlations have been used. In general, this is assumed to have a negative bias on the analysis.
- For every analysis conducted in this study, the fuel was assumed to have a cosine-shaped power profile. The shape was not changed during any transient. In

[c] By 'negative bias' it is therefore implied that the results presented here are less favorable than those that would be expected in the analysis conducted by the fuel manufacturers.

general, this would impart a positive bias on the analysis. The real power profiles used by fuel vendors in licensing-level calculations are proprietary.

- Not all plant uncertainties were considered in this analysis. In particular, no accommodation has explicitly been made for core anomalies or transitional core conditions. No consideration has been given for degradation of fuel thermal performance over the life of the core. The analysis required to accurately predict the proper influence of these uncertainties on core T/H behavior is beyond the scope of this study. These assumptions would positively bias the analysis.
- The true statistical uncertainty associated with plant parameters that were considered in the analysis: pressure, core inlet temperature, mass flux, linear power and $F_{\Delta H}^{E}$ were not accurately described. Rather they were assumed to be a value representative of historical designs. This could provide either a positive or negative bias to the analysis.
- Finally, the analysis conducted is based on design and simulations that are perpetually in a state of refinement. This could provide either a positive or negative bias to the analysis.

Subject to the above qualifications, this analysis has shown that the IRIS core, as designed, provides satisfactory T/H margin to accommodate the LR/SS and PLOFA transients. Referring to the results presented in Table 15 for the W-3L correlation, without the consideration of uncertainty, the IRIS core can also safely cool the core in the event of a CLOFA. Complete acceptability of the IRIS core is not demonstrated for the CLOFA under conditions of design uncertainty.

4.2. Comparison of Hot Channel Analysis Methodology

It should be clear that the STDP provides the most conservative result. Within the assumptions of the T/H analysis, the plant conditions can never be worse than those used for the STDP transient analysis. Therefore, if the plant performs satisfactorily under those circumstances, the plant must be satisfactory for all conditions expected during steady state operations, condition I and condition II transients. It can be seen from the results presented in Table 15 - Table 17 that this is the case.

The ITDP eliminates some of the unnecessary conservatism in two ways. First, the assumption is made that each uncertain parameter may vary independently of the others. This puts a mathematical formalism to the notion that there is a lower probability associated with the condition of all uncertain parameters simultaneously achieving their

most adverse value. Secondly, rather than aiming for the sort of 'absolute' conservatism provided by the STDP, a lower mark is placed at 'acceptable' conservatism defined by the 95% probability with 95% confidence criteria.

One weakness of the ITDP, as already discussed, is the reliance on a single set of values that describes the sensitivity of MDNBR to perturbations in the uncertain parameters. It has been shown that these sensitivity parameters are not, in fact constant. Therefore the requirement to choose a single set of conservative values for these parameters adds undue conservatism to the ITDP. This weakness is a fundamental property of the mathematical formulation of the ITDP, and cannot be avoided.

Another weakness of the ITDP, is its inability to utilize all of the information present in the associated uncertain parameter probability distributions rather than simply the distribution mean and variance. Consider the case of an uncertain parameter described by a rectangular distribution. This distribution has a mean and variance as computed earlier in this chapter. Suppose that some additional information was gained regarding this uncertain parameter such that, instead of a rectangular distribution, the parameter should instead by described by a normal distribution. Suppose further that this normal distribution was determined to have the same mean and variance as the rectangular distribution. A graphical representation of this is provided in Figure 42.

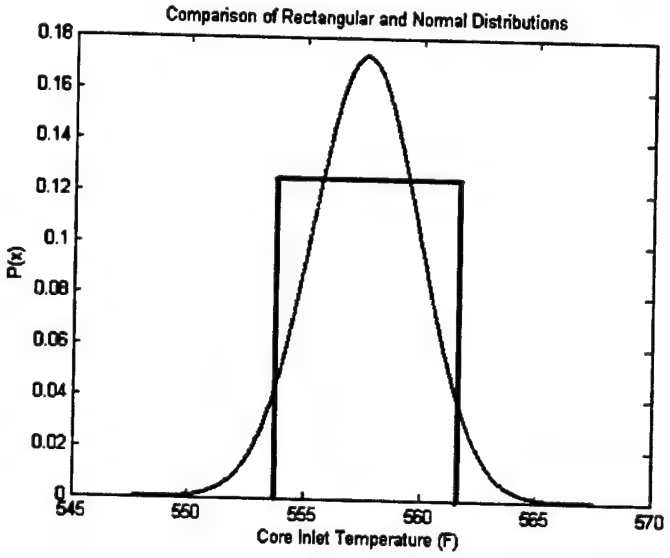


Figure 42: Normal and Rectangular PDFs with identical mean and variance.

What would change in the uncertainty analysis if the ITDP were used? The answer of course is: nothing! The only information that the ITDP uses regarding the probability distribution of the random parameter is its mean and variance. In contrast, the MCUP would automatically take this difference into account – no mathematical meddling required. This change in probability distribution has an impact on the behavior of the uncertain parameter. Since the ITDP is designed to be conservative, even if the change in the shape of the probability distribution were favorable from a T/H standpoint, there would be no mechanism for taking this change into account.

Is this change in the probability distribution of the inlet temperature favorable from a T/H standpoint? In this case, it is clear from the above figure that core inlet temperatures near the mean value would be more likely. This is a favorable feature insofar as it is less likely that the core inlet temperature would be higher than normal. On the other hand, the normal distribution has the feature that there is a small, but finite probability that the inlet temperature would be significantly outside of the previously determined band of ± 4 °F. This is probably an unfavorable feature since, due to the increase in correlation sensitivity as the system approaches CHF (i.e. the core inlet temperature increases) and reduction in sensitivity as the system moves further from CHF, the instances of increased inlet temperature would be more detrimental, from a CHF standpoint, than the benefit obtained from reduced core inlet temperatures. It is not immediately clear whether these two factors would combine to be thermal-hydraulically favorable or unfavorable. It turns out, as reported using the MCUP, that the combined effect is slightly favorable. This can be seen in Table 15 with the Bowring correlation and in Table 16 with the EPRI correlation.

The MCUP has been shown to consistently produce more favorable thermal-hydraulic results with the same level of confidence as the ITDP. To the extent that the assumptions governing the T/H analysis are correct, the MCUP is a more accurate method to quantify core uncertainties.

Chapter 5 Future Work

5.1. Objectives

The objectives of future work stemming from this thesis can be grouped into three categories:

- Thermal Hydraulic Analysis
  - Qualification of appropriate DNB correlation
  - Develop a complete set of transient analyses
  - Benchmark MCUP methodology to proprietary uncertainty analysis procedures.
  - Guidance for Loss of Flow Accident analysis
- Script-based Interface Tools
  - Extension of interface to integrate plant simulation tools such as RELAP.
  - Migration of interface platform to use non-proprietary software tools.
- IRIS Open Economic Analysis[a]

[a] Original work provided in Appendix 9.

5.2. Scope

IRIS Open Thermal Hydraulic Analysis

Qualification of DNB Correlation

A fundamental component to the T/H analysis of any core is the CHF correlation. The quality of this correlation is the cornerstone to the quality of the entire analysis. With this said, the CHF correlations used in this analysis must be improved so as to:

- Be thoroughly tested for validity over the entire range of operational conditions expected for IRIS. In particular, the correlation must be applicable for the relatively low mass-flux conditions that are prevalent for the IRIS reactor; and
- Fully capture the design characteristics of the selected fuel assemblies.

This qualification will require both experimental tests for IRIS specific bundle geometry and operating conditions, and analyses of test results with appropriate VIPRE models to develop an IRIS specific DNB correlation based on other Westinghouse Proprietary Correlation.

Develop Complete Set of Transient Analyses

The various loss of flow accidents that are analyzed in this thesis do not cover the full range of casualties that must be considered for the T/H analysis. The full list transient analyses that will be required for IRIS is described in reference [21].

Benchmark MCUP to Proprietary Uncertainty Analysis Procedures.

The MCUP must be fairly compared against proprietary vendor-designed uncertainty analysis procedures. The MCUP methodology must be expanded to include all parameters that the vendor would include in a full uncertainty analysis, and more precise specification must be given to more accurately and consistently describe the statistical parameters of the uncertain parameters. Upon completion of this task, an evaluation can be made comparing the benefit of using MCUP in relation to the additional computational burden required in doing so.

Guidance for Loss of Flow Accident Analysis

For a typical core design parametric analysis, or core design in which the main objective is not related to thermal hydraulics, it would be useful to have some guidance as to how adequate T/H margin for a LOFA could be reasonably verified without actually conducting the transient analysis. Given the effort that is required to produce accurate analysis results for even a single core design point, such guidance could result in a signficiant reduction in computational requirements and ultimately save considerable time when performing early feasibility studies for a given core design or engaging in broad parametric analysis for a reactor type.

It is suggested that, to account for the T/H degradation that occurs in a LOFA, a steady state analysis could be performed with the nominal coolant flow rate reduced by some percentage, whilst maintaining all other design parameters at their nominal best-guess values. A study should be conducted to determine how large this pre-determined reduction in core flow should be.

Script Interface Tools

Extension of Interface To RELAP

Currently the Matlab-VIPRE interface is deficient in that there is no direct way to incorporate the simulation results of a code such as RELAP to allow for a more seamless transient T/H analysis. A partial implementation of this is provided in Appendix 8. It is recommended that this implementation be completed.

Migration of Script Interface Tools

Probably the biggest disadvantage of the current script interface is that it is implemented using Matlab and, for the Matlab-VIPRE Interface, Microsoft Excel. Though Matlab is a powerful, flexible and user-friendly programming environment, the effective sharing and collaboration with other potential users and developers of the software presented in this thesis may be hindered by the expense, albeit nominal of acquiring Matlab. Presently, free software tools are available, largely useable with the Linux operating system that could supplant proprietary components in every way.

The migration of the script interface software to the Linux (or other POSIX-compliant operating system) would also open the door towards implementing these tools on a parallel distributed platform. As is described in Appendix 2, the sorts of analyses for which the Matlab-VIPRE Interface is most suited can also usually take great advantage of the capabilities of a Beowulf Cluster such as that which is presently available in the Nuclear Engineering Department at MIT. Methodologies such as the MUCP that are at, or beyond, the fringes of practical design feasibility could be made much more practical in this way.

IRIS Open Economic Analysis

A fuel cycle cost analysis has been performed for the IRIS Open core in order to examine the effect of various fuel management strategies on overall fuel cycle costs. Some costs that are not strictly fuel cycle related, namely the cost of replacement electricity during refueling and non-planned forced shutdown as well as O&M costs related to the refueling are also included as refinements of the basic model.

It is recommended that this analysis be refined to more accurately reflect the O&M costs, for which only a rough, order of magnitude approximation was possible. Additionally it is recommended that the Monte Carlo methodology employed to examine the composite effect of uncertainties be refined to more accurately model the uncertainty of all parameters considered by employing more closely tailored probability distributions and distribution parameters.

Chapter 6 Bibliography

[1] BARON, "Risk-Oriented Safety Evaluation of the CAREM-25 Prototype Reactor," *Nucl. Tech.*, **134**:97-109 (2001)

[2] C. STEWART, "VIPRE-01: A Thermal-Hydraulic Code for Reactor Cores. Vol 2 User's Manual," August 1989.

[3] "RELAP5/MOD3.3 Beta Code Manual," NUREG/CR-5535/Rev 1, Inromation Systems Laboratoris, Inc. (2001).

[4] "CASMO-4: A Fuel Assembly Burnup Program," SOA-95/1, Studsvik of America Inc., (1995).

[5] "MCNP (get the rest of this citation)

[6] "FRAPCON-3: A Computer Code for the Calculation of Steady-State, Thermal-Mechanical Behavior of Oxide Fuel Rods for High Burnup," NUREG/CR-6534, PNNL-11513, G. A. BERNA, C. E. BEYER, K. L. DAVIS, D. D. LANNING, Pacific Northwest national Laboratory (1997).

[7] F. BASGMEZ, "Extending a Scientific Application with Scripting Capabilities," *Computing in Science and Engineering*, vol 4, no. 6, pp 52-59.

[8] S. R. BAHN, K. W. JACOBSEN, "An object-oriented scripting interface to a legacy electronic structure code," *Computing in Science and Engineering*, vol 4, no. 3, pp. 56-66.

[9] J. G. B. SACCHERI, "Core Design Strategy for Long-Life Epithermal Water Cooled Reactor," MS Thesis, Massachusetts Institute of Technology, Department of Nuclear Engineering (June 2002).

[10] J. G. B. SACCHERI, "A Tight Lattice, Epithermal, Core Design for the IRIS Reactor: T/H, Neutronics and Economics" PhD Thesis, Massachusetts Institute of Technology, Department of Nuclear Engineering (to be published 2003).

[11] S. LEVY INC., "The Reactor Analysis Support Package (RASP), Volume 1: Introduction and Overview", EPRI NP-4498, (1986)

[12] M. CARELLI , IRIS 2nd Year Technical Report. STD-ES-01-7, March 2002.

[13] L. S. TONG, Y. S. TANG, *Boiling Heat Transfer and Two-Phase Flow*, Taylor & Francis, Washington, DC, 1997.

[14] D. D. Hall, I. MUDAWAR,"Critical Heat Flux(CHF) For Water Flow in Tubes II: Subcooled CHF Correlations," *Int. Jour. Heat and Mass Trans.*, vol. 43, no. 14, pp 2605-2640.

[15] D. D. Hall, I. MUDAWAR, "Critical Heat Flux (CHF) For Water Flow in Tubes I: Compilation and Assessment of World CHF Data," *Int. Jour. Heat and Mass Trans.*, vol. 43, no. 14, pp 2573-2604, July 2000.

[16] P. HEJZLAR, N. E. TODREAS, "Consideration of critical heat flux margin prediction by subcooled or low quality critical heat flux correlations," *Nucl Eng. and Design*, **163**, pp 215-223, 1996.

[17] N. E. TODREAS, M. S. KAZIMI, *Nuclear Systems I: Thermal Hydraulic Fundamentals*, Taylor & Francis, Washington, DC, 1990.

[18] R. B. LAUGHLIN, "The Physical Basis of Computability," *Comp in Sci and Eng.*, vol. 4, no. 3, pp 27-30, May/June 2002.

[19] D. C. GROENVELD et al., "1986 AECL-UO critical heat flux look-up table," *Heat Transfer Engineering*, **7**, 1-2 pp 46-62, 1986.

[20] ANSI N-18.2-1973, "Nuclear Safety Criteria for the Design of Stationary Pressurized Water Reactors," Am. National Standards Inst., Inc., New York.

[21] L. ORIANI, " IRIS Preliminary Safety Assessment," WCAP-16082-NP, (Jul 2003).

[22] H. CHELMER, L. H. BOWMAN, D. R. SHARP, *Improved Thermal Design Procedure.*. Westinghouse Report WCAP-8565, July 1975.

[23] N. E. TODREAS, M. S. KAZIMI, *Nuclear Systems II: Elements of Thermal Hydraulic Design.*, Taylor & Francis, Washington, DC, 2001.

[24] A. H-S. ANG, W. H. TANG, *Probability Concepts in Engineering Planning and Design*, John Wiley & Sons, New York, 1975.

[25] HOGG, TANIS,*Probability and Statistical Inference*, Prentice Hall, Upper Saddle River, NJ, 2001.

[26] R. C. ANDERSEN, K. L. BASHORE,"Monte-Carlo Departure from Nucleate Boiling Ratio Limit Development Using Flow Channel Thermal Hydraulic Code Models," *Nucl. Technol.,* **77**,227 (1987).

[27] S. H. LEE, H. K. KIM,"Development of Statistical Core Thermal Design Methodology Using a Modified Latin Hypercube Sampling Method," *Nucl. Technol.*, **94**, 407 (1991).

[28] L. CINOTTI, F. L. RIZZO, "The Inherently Safe Immersed System (ISIS) Reactor," *Nucl. Eng. Des.*, **143**:295-300 (1993)

[29] M. H. CHANG, S. K. SIM, D. J. LEE, "SMART Behavior Under Over-Pressurizing Accident Conditions," *Nucl. Eng. Des.*, **199**:187-196 (2000)

[30] T. KUSONOKI, "Design of Advanced Integral-type Marine Reactor, MRX," *Nucl. Eng. Des.*, **201**:155-175 (2000)

[31] R. M. KURIDAN, T. D. BEYNON, "Analysis of the Steam Generator for the Safe Integral Reactor Concept: I. Steady State," *Prog. In Nucl. Energy*, **31-3**:273-287 (1997)

[32] A. A. BELYAEV, A. V. KURACHENKOV, Y. K. PANOV, O. B. SAMOILOV, "Nuclear Energy Sources on the Basis of Integral Reactors," *Nucl. Eng. Des.* **173**:131-141 (1997)

[33] O. B. SAMOILOV, V. S. KUUL, V. A. MALAMUD, G. I. TARASOV, "Integral Nuclear Power Reactor with Natural Coolant Circulation: Investigation of Passive RHR System, " *Nucl. Eng. Des.*, **165**:259-264 (1996)

[34] O. B. SAMOILOV, A. V. KURACHENKOV, "Nuclear District Heating Plants AST-500. Present Status and Prospects for Future in Russia," *Nucl. Eng. Des.*, **173**:109-117 (1997)

[35] A. N. ACHKASOV, G. I. GRECHKO, V. A. SHISHKIN, "Nuclear Steam Supply System With an Integral Nuclear Reactor for a Floating NPP," *Nucl. Eng. Des.*, **173**:193-199 (1997)

[36] W. DAZHONG, "The Design Characteristics and Construction Experiences of the 5 MWt Nuclear Heating Reactor, " *Nucl. Eng. Des.*, **143**:19-24 (1993)

[37] W. DAZHONG, Z. DAFANG, D. DONG, G. ZUYING, L. HUAIXUAN, L. JIAGUI, S. QINGSHAN, "Experimental Study and Operation Experiences of the 5 MW Nuclear Heating Reactor," *Nucl. Eng. Des.*, **143**:9-18 (1993)

[38] W. DAZHONG, G. ZUYING, Z. WENXIAN, "Technical Design Features and Safety Analysis of the 200 MWt Nuclear Heating Reactor," *Nucl. Eng. Des.*, **143**:1-7 (1993)

[39] R. D. KLAKE, R. WORSCHECH, "*NS Otto Hahn* Non-Destructive Retesting (in-service inspection)", Symposium on the Safety of Nuclear Ships, p243, OECD, 1978.

[40] R. D. KLAKE, H. KRACHT, "The Consolidated Steam Generator in the Advanced Pressurized Water Reactor," Symposium on the Safety of Nuclear Ships, p607, OECD, 1978.

[41] K. SCHMIDT, D. SEELIGER, H. ZEIBIG, "Engineered Safety Equipment and Safety Analysis of NCS 80," Symposium on the Safety of Nuclear Ships, p265, OECD, 1978.

[42] J. SCHARGE, P. WEHOWSKY, H. ZEIBIG, "In-Service Inspection Program for the NCS-80 Reactor Pressure Vessel," Symposium on the Safety of Nuclear Ships, p285, OECD, 1978.

[43] R. F. POCOCK, *Nuclear Ship Propulsion*, Ian Allan Ltd., London, UK 1970.

[44] "CNSG Development and Status: 1969-1977," Babcock & Wilcox, BAW-1454, 1978.

[45] "Steam: Its Generation and Use," Babcock & Wilcox, 1978.

[46] "Small Nuclear Power Plants: Vol 2 The Industrial Expression of Supply and Demand Considerations, COO-284 (Vol. 2), Chicago Operations Office of the U. S. Atomic Energy Commission, (Mar 1967).

[47] D. HANSELMAN, B. LITTLEFIELD, *Mastering Matlab 6: A Comprehensive Tutorial and Reference*, Chap 34, p. 561-602, Prentice Hall, Upper Saddle River, New Jersey (2001)

[48] C STEWART, "VIPRE-01: A Thermal-Hydrauic Code for Reactor Cores. Vol 3 Programmer's Manual," July 1985.

[49] OECD/NEA, "International Nuclear Fuel Cycle Cost Evaluation," IAEA, ISP534, vol. 1-9 (1980)

[50] OECD/NEA, "The Economics of the Nuclear Fuel Cycle," (1994)

[51] W. I. KO, H. CHOI, and M. S. YANG, "Economic Analysis on Direct Use of Spent Pressurized Water Reactor Fuel In CANDU Reactors – IV: DUPIC Fuel Cycle Cost," *Nucl. Tech.*, **vol** 134, (May 2001)

[52] M. R. GALVIN, N. E. TODREAS, "Maintenance Cycle Extension in Advanced Light Water Reactor Design," MIT-NSP-TR-004, October 2001.

Chapter 7 Appendices

Appendix 1 Integral Reactor Database

The following is a listing of Integral Reactor Projects.

Recent and Ongoing Projects

The following data has been collected from open literature. Blank entries indicate that the data was unavailable or not found.

Design (Associated References)	Power (MWth)	Primary Pressure (MPa)	Primary Temp (in/out) - °C	Secondary Pressure (MPa)	Integral Configuration		
					SGs	RCPs	Pressurizer
IRIS [12]	1000	15.5	292/328	7	8 helical coil OTSG	8 – above SGs	Upper Head
ISIS [28]	650	14	271/310	4.6	helical coil OTSG	2- above SG	External to RPV
SMART [29]	330	15	270/310	3	12 helical coil OTSG	4 – mounted on RPV Head	Upper Head
CAREM-25 [1]	~75 (25MWe)				12 OTSG		Upper Head
MRX [30]	100	12	283/298	4	Inconel 800 OTSG	External to core	Upper Head

SIR [31]	1000	15.5	295/318	5.5	12 Straight tube OTSG	Around upper circumference of vessel.	Upper Head
ATEC-200[a] [32],[33]	700	15.7	?/343	4.4	Titanium OTSG	Natural Circulation	Upper Head –Loaded with Nitrogen (?)
AST-500 [32],[34]	500	1.96	134/208	1.2		Natural Circulation	Steam/gas integral
VPBER-600 [32]	1800	15.7	?/325	6.4			
Floating NPP [35]	2x42	15.7	265/310	3.5			
NHR-5[b] [36]-[38]	5	1.5	146/186	1.7		Natural Circulation	

Old Projects

Design (Associated References)	**Power (MWth)**	**Primary Pressure (MPa)**	**Primary Temp (in/out) - °C**	**Secondary Pressure (MPa)**	**Integral Configuration**		
					SGs	**RCPs**	**Pressurizer**

[a] In reference [32], the authors note that the ATEC-200: "...was developed based on design and construction experience obtained with the pilot power units AST-500, and the proven design solutions and established technologies of nuclear-powered icebreakers, whos operating performances have been confirmed by about 40 years of successful operations in Arctic seas." This is a standardized series of designs: ATEC-80,-150,-200.

[b] A larger 200 MWth plant is to be (or has been) built based on the positive experiences and lessons learned from this project. Though some of the operational characteristics of this plant are obviously fundamentally different from the IRIS, the concept of an integral SG was successfully employed.

Otto Hahn [39],[40]	38	912 psi (fix)		523 psi	Helical tube OTSG	Mounted External	Upper Head
NCS-80 [41],[42]							
CNSG [43]-[45]	38	918 psia	523/? (F)	440 psia	12 straight-tube OTSG	Mounted on RPV head	
UNIMOD [46]	80	2000-2500 psia		640 psia	6 U-shaped tube OTSG	Mounted Astride of RPV	
U1U/U2U [46]	16.5/51.6 MWe	1050 psia	515/545 (F)	415 psia	3 OTSG	Mounted Astride of RPV	Integral
Vulcain [43]							

Appendix 2 Matlab-VIPRE Interface

In this section, the code developed for the Matlab-VIPRE Interface is presented and explained to a greater depth. The appendix is broken down to describe the major features.

Utility Functions

The Matlab-VIPRE Interface as described in Chapter 2, requires extensive communication between Microsoft Excel, the running scripts within the Matlab Workspace context, the file system for input and output files, and the operating system to initiate a particular VIPRE execution.

These functions are short, but represent critical links between the rather routine and tedious functions required for input data extraction, input file generation and overall execution coordination.

Communication Between Matlab and Excel

An absolutely essential ingredient to the Matlab-VIPRE Interface is the communication pathway between Matlab and Excel. This communication is done through Activex objects. Activex is the Microsoft implementation of the COM protocol for inter-process communication. Excel is run as an Activex Server, Matlab is the Activex Client. Communication is achieved through function calls from the client to the server.

The first function presented is: get_spreadsheet.m. This function takes as an argument a fully qualified path name to an existing Microsoft Excel workbook. The function returns two objects: first, a handle to the Excel application Activex Object,[a] the second is a handle to the specific workbook identified by the input argument.

```
%get_spreadsheet.m
%function [excel_actx_obj,spread_sheet] = get_spreadsheet(file_name) takes as arguements
%a string literal that is the file-name of the desired spreadsheet.  The return values
```

[a] This is just an interface to the Microsoft Excel application. All inter-process communication functionality ultimately is derived from this object.

```
%are active-x objects:  a spreadsheet that must ultimately be closed via a call to:
% invoke(spread_sheet,'Close').  The excel_actx_obj must be deleted via a call to:
% delete(excel_actx_obj);

%an excel.Application that must be  deleted with a call such as:
%delete(excel_actx_obj)
function [excel_actx_obj,spread_sheet] = get_spreadsheet(file_name)

excel_actx_obj = actxserver('Excel.Application');
xl_wb = get(excel_actx_obj,'Workbooks');%creates a handle to the application objects
workbooks
spread_sheet =  invoke(xl_wb,'Open',file_name);
```

The spread_sheet object returned from the get_spreadsheet function does not directly provide access to data within any individual worksheet[b]. To do so, one must obtain a handle to a particular worksheet object. The function get_worksheet.m is provided for this purpose.

This function accepts as arguments the handle to the Excel workbook in which the worksheet resides, as well as a character string that identifies the name of the worksheet. The function returns a handle to the specific worksheet within the workbook.

```
%get_worksheet.m
%takes as arguements the handle to a workbook and a string giving the name of the desired
worksheet
%returns a handle to the named worksheet.

function input_sheet = get_worksheet(workbook,sheet_name)

input_sheet = get(workbook,'Sheets',sheet_name); %get the input sheet
```

The actual data from a particular worksheet may now be accessed through the handle to the worksheet object. To do so, requires a function call through this object. Wrapper functions were written to make this task more intuitive and less error-prone. Two versions are used in the input-data extraction scripts they are: get_cell_value.m and get_sheet_row_col.m.

The function get_cell_value.m accepts as arguments the handle to the specific worksheet that data is being accessed from, and a character string representing the specific worksheet cell that is desired.[c] The return value is the numerical contents of the cell[d].

---

[b] For clarity: A 'workbook' is a generic name for a saved Microsoft Excel file. A 'spreadsheet' is a particular Microsoft Excel spreadsheet. A spreadsheet is composed of one or more worksheets. The input data is stored on a particular worksheet.

[c] e.g. 'A1' or 'B2'.

[d] That is – the formulae upon which the displayed numerical cell value is based are not returned. For example, if cell B2 is 'programmed' within the spreadsheet to be equal to the sum of cells A1 and A2, and

```
%get_cell_value.m

function value = get_cell_value(ws_handle,cell_str)

range = get(ws_handle,'Range',cell_str,cell_str);
form = get(range,'Formula');
if isempty(form)
    value = 'empty'; %default
else
    value = get(range,'Value');
end
```

The function get_sheet_row_col.m works in a similar fashion, but accepts as arguments, a handle to the desired worksheet, and two integers representing the specific row and column for which the value is desired. This variant was chosen to allow loop indexing within the function call.[e] The code is as follows:

```
%get_sheet_row_col.m

function val = get_sheet_row_col(ws_handle,row,column)

cell = get(ws_handle,'Cells',row,column);
form = get(cell,'Formula');
if isempty(form)
    val = 'empty';
else
    val = get(cell,'Value');
end
```

Initiation of VIPRE execution

When the input file has been generated and all is ready for the commencement of a VIPRE run, there must be some way in which Matlab can cause VIPRE to begin running. This sort of functionality is normally made through function calls through the operating

---

the resulting sum is 2, then 2 is returned, not the expression A1+A2. If the cell is completely empty, a character string 'empty' is returned. Normally, when the input extraction scripts are written, it is not expected that any of the cells would be empty. The VIPRE input manual allows for default values however and the input-generation must account for the fact that some cells may be empty, and deal with that situation appropriately, either by inserting the default value into the input file, or leaving a blank space delimited by a comma. The 'empty' return value is provided so that the input generation script can distinguish between a variable that should actually have the value '0', and one that was merely not entered, and the default value should be used instead (clearly, not necessarily 0).

[e] I.e. There is no simple way to automatically loop from A1 to A2 or A1 to B1 in a looping type construct within the Matlab language. It is desirable in this respect to have integers represent the row and column of the desired data so that one can easily loop of a series of rows or columns for a series of input data values. Additionally, it is useful to allow some flexibility regarding the placement of the input data within a specific spreadsheet. If, for example, the input model is changed by un-lumping a collection of channels – more channel types may be necessary, this could, depending on the organization of the spreadsheet used to hold the input data, offset the spreadsheet position of many other bits of input data. If an integer index variable is used – the portion of the input extraction program can be updated much more quickly and easily to reflect a change in position of the input data, than one could if one were using the semantics required in the get_cell_value.m function.

system itself. In Matlab Student version 5.3, for which this interface was written, there is no capability of making operating system calls directly.[f] Despite this, Matlab is endowed with the capability of being extended through C or Fortran functions. These functions are ordinary C or Fortran subroutines[g] with slight formatting modifications to allow for passage of input and output variables from within the Matlab workspace context to the memory space allotted for the function. These functions are colloquially referred to as "mex-files" because the C or Fortran compiler used is called through the Matlab function mex (with appropriate arguments). A more complete description with more general examples of this capability is given in reference [47]. The source code is presented here:

```
#include "mex.h"
#include <stdlib.h>
#include <stdio.h>

void mexFunction(int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[])
{
  printf("\nCalling Vipre...\n ");
            system("vipre");
            return;
}
```

[f] As in, through a DOS command line or UNIX shell command. In the newest student release of Matlab, Version 6.5, this functionality is provided.

[g] Though these 'subroutines' can have nearly arbitrary complexity and may invoke any number of other subroutines or executable programs, so there is no implication here that the 'subroutines' have to be simple.

Input Data Extraction

The script employed for input data extraction is an exceptionally fragile portion of the interface. Every core model created will require a different script to properly extract the data. What is uniform from one input-data-extraction script to the next, are the variables in which the data is placed, and the manner in which the data are placed there. Additionally, each data extraction script used for this research was based on the fact that all of the required input data resides in a Microsoft Excel spreadsheet.

For the research described here, dozens of different input-data-extraction scripts have been used. Presented here is an example of these scripts that demonstrates all of the required characteristics of these scripts. Any researcher wishing to make use of the Matlab-VIPRE Interface described in this thesis will need to understand how this script works to the degree that he or she will be able to create a new script of his or her own.

This particular script is used to extract input data to run a transient simulation using the W-3L correlation. The difference in the input-data-extraction scripts required for transients rather than steady state lie in the requirement to extract the large table of core boundary condition data.

Source Code:

This function requires the following inputs:

- input_sheet: This is an Activex handle to the Excel Spreadsheet that holds all of the input data other than the core boundary condition data required for the transient.
- trans_input_sheet: This is an Activex handle to the Excel Spreadsheet that holds all of the core boundary condition data required for the transient.
- var_file_name: This is a character string specifying the required name for *.mat file that will hold the input data.

The effect of the function is to place a file[h] within the current working directory that contains (in an easily accessible form[i]) all of the input data required to generate a valid VIPRE input file.

```
%get_excel_input_trans_w_31.m

function get_excel_input_trans_w_31(input_sheet,trans_input_sheet,var_file_name)


vipre1.kase = get_cell_value(input_sheet,'A32');
vipre1.irstrt = get_cell_value(input_sheet,'B32');
vipre1.irstep = get_cell_value(input_sheet,'C32');
output_que = char('vipre1');

vipre2.text = get_cell_value(input_sheet,'A33');
output_que = char(output_que,'vipre2');

geom1.inflag = 'geom';
geom1.nchan1 = get_cell_value(input_sheet,'B40');
geom1.ncard = get_cell_value(input_sheet,'C40');

if geom1.ncard == 0
    compressed = 1;
else
    compressed = 0;
end

geom1.ndx = get_cell_value(input_sheet,'D40');
geom1.nazone = get_cell_value(input_sheet,'E40');
if geom1.nazone == 0
    uniform_axial_nodes = 1;
else
    uniform_axial_nodes = 0;
end

geom1.nctyp = get_cell_value(input_sheet,'F40');
geom1.mbwr = get_cell_value(input_sheet,'G40');
output_que = char(output_que,'geom1');

%start geom2

if geom1.ndx > 0
    %read geom2 starting with cell a42
    geom2.zz = get_cell_value(input_sheet,'A42');
    geom2.theta = get_cell_value(input_sheet,'B42');
    geom2.sl = get_cell_value(input_sheet,'C42');
    output_que = char(output_que,'geom2');
end

if geom1.nazone > 0
    index = 0;
    for i = 1:geom1.nazone
        index = index + 1;
        geom3(i).dx = get_sheet_row_col(input_sheet,44,index);
        index = index+1;
        geom3(i).xboun = get_sheet_row_col(input_sheet,44,index);
    end
    output_que = char(output_que,'geom3');
end

if compressed == 1
```

[h] This file will have the name *.mat, where the * is replaced by the character string represented by the input variable var_file_name. (e.g. where var_file_name = 'run1', the file will be called run1.mat)

[i] From the previous footnote, to place all of the data from this file into the Matlab workspace context, simply give the command: load run1

```
        %compressed geometry is used
        %first geom5 entry is in row 47
        start_index = 47;
        for i = 1:geom1.nctyp
            index = start_index + (i-1)*4;
           geom5(i).mchn = get_sheet_row_col(input_sheet,index,1);
           geom5(i).carea = get_sheet_row_col(input_sheet,index,2);
           geom5(i).cpw = get_sheet_row_col(input_sheet,index,3);
           geom5(i).cph = get_sheet_row_col(input_sheet,index,4);
           index = index + 2;
           for j = 1:geom5(i).mchn
               geom6(i,j) = get_sheet_row_col(input_sheet,index,j);
           end

       end
       output_que = char(output_que,'geom5','geom6','geom7');
       %geom7 cell start is A96

       geom7.nk = get_cell_value(input_sheet,'A96');
       geom7.ngtyp = get_cell_value(input_sheet,'B96');
       geom7.ngap = get_cell_value(input_sheet,'C96');
       geom7.nrow = get_cell_value(input_sheet,'D96');
       geom7.isym = get_cell_value(input_sheet,'E96');
       geom7.nsrow = get_cell_value(input_sheet,'F96');

       if (geom7.nk > 0) & (geom7.ngap == 1)
           %this means that we read geom11 -- geom11 starts on line 'A99'
           start_index = 99;
           for i = 1:geom7.nk
               index = start_index + (i-1);
               geom11(i).k = get_sheet_row_col(input_sheet,index,1);
               geom11(i).I = get_sheet_row_col(input_sheet,index,2);
               geom11(i).J = get_sheet_row_col(input_sheet,index,3);
               geom11(i).width = get_sheet_row_col(input_sheet,index,4);
               geom11(i).cent = get_sheet_row_col(input_sheet,index,5);
           end
           output_que = char(output_que,'geom11');
       end

end

%starting row is:A132
prop1.inflag = 'prop';
prop1.nprop = get_cell_value(input_sheet,'B132');
prop1.isteam = get_cell_value(input_sheet,'C132');
prop1.nfprop = get_cell_value(input_sheet,'D132');
prop1.ipvar = get_cell_value(input_sheet,'E132');
output_que = char(output_que,'prop1');

%rods control group -- starts at cell A135
rods1.inflag = 'rods';
rods1.naxp = get_cell_value(input_sheet,'B135');
rods1.nrod = get_cell_value(input_sheet,'C135');
rods1.nc = get_cell_value(input_sheet,'D135');
rods1.nfuelt = get_cell_value(input_sheet,'E135');
rods1.nmat = get_cell_value(input_sheet,'F135');
rods1.igpff = get_cell_value(input_sheet,'G135');
rods1.ngpff = get_cell_value(input_sheet,'H135');
rods1.nopt = get_cell_value(input_sheet,'I135');
rods1.ipowv = get_cell_value(input_sheet,'J135');
rods1.icpr = get_cell_value(input_sheet,'K135');
rods1.irff = get_cell_value(input_sheet,'L135');
output_que = char(output_que,'rods1');

%rods2 starts at cell A137
rods2.zzh = get_cell_value(input_sheet,'A137');
rods2.zstrt = get_cell_value(input_sheet,'B137');
rods2.nodals = get_cell_value(input_sheet,'C137');
rods2.nodalt = get_cell_value(input_sheet,'D137');
output_que = char(output_que,'rods2');
```

```
%rods3 starts at cell A139
rods3.naxn = get_cell_value(input_sheet,'A139');
output_que = char(output_que,'rods3');

%rods5 starts at cell A141
rods5.pstar = get_cell_value(input_sheet,'A141');
output_que = char(output_que,'rods5');

%rods9 starts at cell A143
if rods1.nopt == 0
    start_index = 143;
    for i = 1:rods1.nrod
        index = start_index + (i-1);
        rods9_pre(i).I = get_sheet_row_col(input_sheet,index,1);
        rods9_pre(i).idfuel = get_sheet_row_col(input_sheet,index,2);
        rods9_pre(i).radial = get_sheet_row_col(input_sheet,index,3);
        rods9_pre(i).iaxp = get_sheet_row_col(input_sheet,index,4);

        j = 1;%loop index
        cont = 1;%logical variable for loop
        %fill remainder of values for rods9 -- This is more difficult
        %since I don't assume knowledge of the number of channels facing
        %each rod -- so I have to determine this on the fly.
        %that is the purpose of the following logic
        while(cont == 1)
            next_val = get_sheet_row_col(input_sheet,index,4+j);
            if (next_val == 'empty')
                cont = 0;
                rods9(i,j) = 0;
                continue
            else
                rods9(i,j) = next_val;
            end
            j = j+1;
        end

    end
end
output_que = char(output_que,'rods9');

%must remember to terminate this list with a zero when generating the input file

%rods.68 input -- for no fuel conduction model
row_index = 169;
for i = 1:rods1.nfuelt
    rods68(i).I = get_sheet_row_col(input_sheet,row_index+(i-1),1);
    rods68(i).ftype = get_sheet_row_col(input_sheet,row_index+(i-1),2);
    rods68(i).drod = get_sheet_row_col(input_sheet,row_index+(i-1),3);
    rods68(i).dfuel = get_sheet_row_col(input_sheet,row_index+(i-1),4);
    rods68(i).nfuel = get_sheet_row_col(input_sheet,row_index+(i-1),5);
end
output_que = char(output_que,'rods68');

%================ Input section for fuel conduction model==============

%ADD RODS.62 input here
%rods62 starts on cell A169
%rods62.I = get_cell_value(input_sheet,'A169');
%rods62.ftype = 'nucl';
%rods62.drod = get_cell_value(input_sheet,'c169');
%rods62.dfuel = get_cell_value(input_sheet,'d169');
%rods62.nfuel = get_cell_value(input_sheet,'e169');
%rods62.dcore = get_cell_value(input_sheet,'f169');
%rods62.tclad = get_cell_value(input_sheet,'g169');
%output_que = char(output_que,'rods62');

%ADD RODS.63 input here
%rods63 starts on cell A171
%rods63.iradp = get_cell_value(input_sheet,'A171');
%rods63.imatf = get_cell_value(input_sheet,'B171');
%rods63.imatc = get_cell_value(input_sheet,'C171');
```

```
%rods63.igpc = get_cell_value(input_sheet,'D171');
%rods63.igforc = get_cell_value(input_sheet,'E171');
%rods63.hgap = get_cell_value(input_sheet,'F171');
%rods63.ftdens = get_cell_value(input_sheet,'G171');
%rods63.fclad = get_cell_value(input_sheet,'H171');
%output_que = char(output_que,'rods63');

%ADD RODS.64 input here
%start_index = 173; %input starts on row 173
%cont = 1;
%    count = 0;
%    i = 0;
%    while(cont == 1)
%        for j = 1:8 %maximum number of entries in a row in the spreadsheet
%            count = count + 1;
%            if (count <= 2*rods63.iradp)
%                rods64(count) = get_sheet_row_col(input_sheet,start_index+i,j);
%            else
%                cont = 0;
%            end
%            if (cont == 0)
%                j = 9;
%                break;
%            end

%        end
%        i = i+1; %move to the next row
%    end
%output_que = char(output_que,'rods64');
%================================================================================

%oper.1 input
%starts on row 177
start_index = 177;
oper1.inflag = get_sheet_row_col(input_sheet,start_index,1);
oper1.ih = get_sheet_row_col(input_sheet,start_index,2);
oper1.ig = get_sheet_row_col(input_sheet,start_index,3);
oper1.isp = get_sheet_row_col(input_sheet,start_index,4);
oper1.npowr = get_sheet_row_col(input_sheet,start_index,5);
oper1.ndnb = get_sheet_row_col(input_sheet,start_index,6);
oper1.irun = get_sheet_row_col(input_sheet,start_index,7);
oper1.ifcvr = get_sheet_row_col(input_sheet,start_index,8);
oper1.luf = get_sheet_row_col(input_sheet,start_index,9);
oper1.ihbal = get_sheet_row_col(input_sheet,start_index,10);
output_que = char(output_que,'oper1');

%oper.2 input
%starts on row 179
start_index = 179;
oper2.dps = get_sheet_row_col(input_sheet,start_index,1);
oper2.dnbrl = get_sheet_row_col(input_sheet,start_index,2);
oper2.fcool = get_sheet_row_col(input_sheet,start_index,3);
oper2.dnbrc = get_sheet_row_col(input_sheet,start_index,4);
oper2.ihrod = get_sheet_row_col(input_sheet,start_index,5);
output_que = char(output_que,'oper2');

%oper.5 input
%starts on row 181
start_index = 181;
oper5.pref = get_sheet_row_col(input_sheet,start_index,1);
oper5.hin = get_sheet_row_col(input_sheet,start_index,2);
oper5.gin = get_sheet_row_col(input_sheet,start_index,3);
oper5.pwrinp = get_sheet_row_col(input_sheet,start_index,4);
oper5.hout = get_sheet_row_col(input_sheet,start_index,5);
output_que = char(output_que,'oper5');

%oper.11 input

if (oper1.isp == 1) | (oper1.isp == 2)
    start_index = 183;
    %condition for using oper.11
```

```
    %starting row is 183
    cont = 1;
    count = 0;
    i = 0;
    while(cont == 1)
        for j = 1:8 %maximum number of entries in a row in the spreadsheet
            count = count + 1;
            if (count <= geom1.nchan1)
                oper11.flofrc(count) = get_sheet_row_col(input_sheet,start_index+i,j);
            else
                cont = 0;
            end
            if (cont == 0)
                j = 9;
                break;
            end

        end
        i = i+1; %move to the next row
    end
end
output_que = char(output_que,'oper11');


%oper.12 input
%begins on row 187
start_index = 187;
oper12.np = get_sheet_row_col(input_sheet,start_index,1);
oper12.nh = get_sheet_row_col(input_sheet,start_index,2);
oper12.ng = get_sheet_row_col(input_sheet,start_index,3);
oper12.nq = get_sheet_row_col(input_sheet,start_index,4);
oper12.nnath = get_sheet_row_col(input_sheet,start_index,5);
oper12.nnatg = get_sheet_row_col(input_sheet,start_index,6);
output_que = char(output_que,'oper12');


%=============================================================================================
%=================== Get Transient Data ======================================================
%=============================================================================================

%here, the transient forcing functions need to be read off of the trans_input_sheet.

%oper13 -- will be stored as an array. - System pressure forcing function
%the first pair of values will be the initial condition
oper13 = zeros(abs(oper12.np)*2,1); %a vector containing all of the values
oper14 = oper13;
oper17 = oper13;
oper20 = oper13;
%oper13(1) is time zero, and need not be modified
%oper13(2) is the time zero pressure and is given
oper13(2) = oper5.pref;

power_col_index = 14;
flow_col_index = 16;
time_col_index = 12;
press_col_index = 15;
temp_col_index = 17;
%oper14 -- temperature forcing function
oper14(2) = oper5.hin;
%oper17 -- inlet flow forcing function
oper17(2) = oper5.gin;
%oper20 -- power forcing function
oper20(2) = oper5.pwrinp;

time_index = 2;
for i = 3:2:(abs(oper12.np*2))
    oper13(i) = get_sheet_row_col(trans_input_sheet,time_index,time_col_index);
    oper13(i+1) = get_sheet_row_col(trans_input_sheet,time_index,press_col_index);
    oper14(i) = oper13(i);
    oper17(i) = oper13(i);
    oper20(i) = oper13(i);
```

```
    oper14(i+1) = get_sheet_row_col(trans_input_sheet,time_index,temp_col_index);
    oper17(i+1) = get_sheet_row_col(trans_input_sheet,time_index,flow_col_index);
    oper20(i+1) = get_sheet_row_col(trans_input_sheet,time_index,power_col_index);
    time_index = time_index + 1;
end

output_que = char(output_que,'oper13');
output_que = char(output_que,'oper14');
output_que = char(output_que,'oper17');
output_que = char(output_que,'oper20');

%=============================================================================================
%=============================================================================================

%group corr begins on row 193
start_index = 193;
corr1.inflag = 'corr';
corr1.ncor = get_sheet_row_col(input_sheet,start_index,2);
corr1.nhtc = get_sheet_row_col(input_sheet,start_index,3);
corr1.ixchf = get_sheet_row_col(input_sheet,start_index,4);
output_que = char(output_que,'corr1');

%corr.2 begins on row 195
start_index = 195;
corr2.nscvd = get_sheet_row_col(input_sheet,start_index,1);
corr2.nblvd = get_sheet_row_col(input_sheet,start_index,2);
corr2.nfrml = get_sheet_row_col(input_sheet,start_index,3);
corr2.nhtwl = get_sheet_row_col(input_sheet,start_index,4);
output_que = char(output_que,'corr2');

%corr.6 begins on row 197
start_index = 197;
corr6.nfcon = get_sheet_row_col(input_sheet,start_index,1);
corr6.nsubc = get_sheet_row_col(input_sheet,start_index,2);
corr6.nsatb = get_sheet_row_col(input_sheet,start_index,3);
corr6.nchfc = get_sheet_row_col(input_sheet,start_index,4);
corr6.ntrnb = get_sheet_row_col(input_sheet,start_index,5);
corr6.nflmb = get_sheet_row_col(input_sheet,start_index,6);
output_que = char(output_que,'corr6');

%corr.7 given on row 199
if corr6.nfcon == 'epri'
    start_index = 199;
    corr7.cdb = get_sheet_row_col(input_sheet,start_index,1);
    output_que = char(output_que,'corr7');
end

%corr.9 given on row 201
start_index = 201;
for i = 1:corr1.ncor
    corr9.nchf(i,:) = get_sheet_row_col(input_sheet,start_index,i);
end
output_que = char(output_que,'corr9');

%corr.11
start_index = 203;
corr11.tdcl = get_sheet_row_col(input_sheet,start_index,1);
corr11.spk = get_sheet_row_col(input_sheet,start_index,2);
corr11.flgrd = get_sheet_row_col(input_sheet,start_index,3);
output_que = char(output_que,'corr11');


%=============================================================================
%=============================================================================
% corr.11 and corr.17 commented out due to removal of W-3L and
%  Macbeth DNB correlation from the analysis.  This input is no
% longer needed.  If I choose to again analyse using these correlations,
% I will need to re-adjust the input spreadsheet and this file
% --- also to be adjusted is the parsing routine to account for the
% reduced number of CHF corellations.
%=============================================================================
```

```
%=====================================================================

%corr.16 given on row 205
%start_index = 205;
%corr16.kbwr = get_sheet_row_col(input_sheet,start_index,1);
%corr16.nuc = get_sheet_row_col(input_sheet,start_index,2);
%corr16.cgrid = get_sheet_row_col(input_sheet,start_index,3);
%output_que = char(output_que,'corr16');

%corr.17 given on row 207
%start_index = 207;
%corr17.mac = get_sheet_row_col(input_sheet,start_index,1);
%output_que = char(output_que,'corr17');

%=====================================================================
%=====================================================================
%=====================================================================

%mixing group starts on row 209 with mixx.1
start_index = 209;
mixx1.nflag = 'mixx';
mixx1.nscbc = get_sheet_row_col(input_sheet,start_index,2);
mixx1.nbbc = get_sheet_row_col(input_sheet,start_index,3);
mixx1.mixk = get_sheet_row_col(input_sheet,start_index,4);
output_que = char(output_que,'mixx1');

%mixx.2 is given on row 211
start_index = 211;
mixx2.ftm = get_sheet_row_col(input_sheet,start_index,1);
mixx2.abeta = get_sheet_row_col(input_sheet,start_index,2);
mixx2.bbeta = get_sheet_row_col(input_sheet,start_index,3);
output_que = char(output_que,'mixx2');

if mixx1.mixk == 1 %then read mixx.3
%mixing parameters given for each gap
%number of gaps is given by geom7.nk
start_index = 213;
    %starting row is 213
    cont = 1;
    count = 0;
    i = 0;
    while(cont == 1)
        for j = 1:8 %maximum number of entries in a row in the spreadsheet
            count = count + 1;
            if (count <= 2*geom7.nk)
                mixx3(count) = get_sheet_row_col(input_sheet,start_index+i,j);
            else
                cont = 0;
            end
            if (cont == 0)
                j = 9;
                break;
            end

        end
        i = i+1; %move to the next row
    end
    output_que = char(output_que,'mixx3');
end

%drag group starts on row 222
start_index = 222;
drag1.inflag = 'drag';
drag1.nchtp = get_sheet_row_col(input_sheet,start_index,2);
drag1.ngptp = get_sheet_row_col(input_sheet,start_index,3);
drag1.kijopt = get_sheet_row_col(input_sheet,start_index,4);
output_que = char(output_que,'drag1');

if drag1.nchtp > 0
    %drag.2 starts on row 224
    start_index = 224;
```

```
    drag2.atf = get_sheet_row_col(input_sheet,start_index,1);
    drag2.btf = get_sheet_row_col(input_sheet,start_index,2);
    drag2.ctf = get_sheet_row_col(input_sheet,start_index,3);
    drag2.alf = get_sheet_row_col(input_sheet,start_index,4);
    drag2.blf = get_sheet_row_col(input_sheet,start_index,5);
    drag2.clf = get_sheet_row_col(input_sheet,start_index,6);
    output_que = char(output_que,'drag2');
end

if drag1.kijopt > 2
    %drag.7 starts on row 226
    start_index = 226;
    drag7.ddok = get_sheet_row_col(input_sheet,start_index,1);
    drag7.ppitch = get_sheet_row_col(input_sheet,start_index,2);
    output_que = char(output_que,'drag7');
end

if drag1.ngptp > 0
    %drag.8 starts on row 228
    start_index = 228;
    drag8.atg = get_sheet_row_col(input_sheet,start_index,1);
    drag8.btg = get_sheet_row_col(input_sheet,start_index,2);
    drag8.ctg = get_sheet_row_col(input_sheet,start_index,3);
    drag8.alg = get_sheet_row_col(input_sheet,start_index,4);
    drag8.blg = get_sheet_row_col(input_sheet,start_index,5);
    drag8.clg = get_sheet_row_col(input_sheet,start_index,6);
    output_que = char(output_que,'drag8');
end

%grids control group starts in row 233

grid1.inflag = 'grid';
start_index = 233;
grid1.kopt = get_sheet_row_col(input_sheet,start_index,2);
grid1.nkcor = get_sheet_row_col(input_sheet,start_index,3);
output_que = char(output_que,'grid1');

%grids.2 given in row 237
start_index = 237;
if grid1.kopt == 0
    for i = 1:grid1.nkcor
     grid2(i) = get_sheet_row_col(input_sheet,start_index,i);
    end
    output_que = char(output_que,'grid2');
end

%grid.4 given in row 239
start_index = 239;
grid4.nci = get_sheet_row_col(input_sheet,start_index,1);
grid4.nlev = get_sheet_row_col(input_sheet,start_index,2);
output_que = char(output_que,'grid4');

%grid.6 entries start on row 241
    start_index = 241;
    cont = 1;
    count = 0;
    i = 0;
    while(cont == 1)
        for j = 1:8 %maximum number of entries in a row in the spreadsheet
            count = count + 1;
            if (count <= (2*grid4.nlev))
                grid6(count) = get_sheet_row_col(input_sheet,start_index+i,j);
            else
                cont = 0;
            end
            if (cont == 0)
                j = 9;
                break;
            end

        end
```

```
        i = i+1; %move to the next row
    end
    output_que = char(output_que,'grid6');

%must terminate the output with a blank entry for grid.4 -- 0 when generating the input
file

%cont group begins on row 249

cont1.inflag = 'cont';
output_que = char(output_que ,'cont1');

%cont.2 begins on row 251
start_index = 251;
cont2.ttdum = get_sheet_row_col(input_sheet,start_index,1);
cont2.ndtdum = get_sheet_row_col(input_sheet,start_index,2);
cont2.ntrydm = get_sheet_row_col(input_sheet,start_index,3);
cont2.itry = get_sheet_row_col(input_sheet,start_index,4);
cont2.itrym = get_sheet_row_col(input_sheet,start_index,5);
cont2.idrect = get_sheet_row_col(input_sheet,start_index,6);
cont2.itstep = get_sheet_row_col(input_sheet,start_index,7);
cont2.itmod  = get_sheet_row_col(input_sheet,start_index,8);
output_que = char(output_que ,'cont2');

%cont.3 begins on row 253
start_index = 253;
cont3.werrx = get_sheet_row_col(input_sheet,start_index,1);
cont3.werry = get_sheet_row_col(input_sheet,start_index,2);
cont3.ferror = get_sheet_row_col(input_sheet,start_index,3);
cont3.terror = get_sheet_row_col(input_sheet,start_index,4);
cont3.htcerr = get_sheet_row_col(input_sheet,start_index,5);
cont3.dampng = get_sheet_row_col(input_sheet,start_index,6);
cont3.accely = get_sheet_row_col(input_sheet,start_index,7);
cont3.accelf = get_sheet_row_col(input_sheet,start_index,8);
output_que = char(output_que, 'cont3');

%cont.6 begins on row 255
start_index = 255;
cont6.nout = get_sheet_row_col(input_sheet,start_index,1);
cont6.npchan = get_sheet_row_col(input_sheet,start_index,2);
cont6.npgap = get_sheet_row_col(input_sheet,start_index,3);
cont6.nprod = get_sheet_row_col(input_sheet,start_index,4);
cont6.npchf = get_sheet_row_col(input_sheet,start_index,5);
cont6.npwl = get_sheet_row_col(input_sheet,start_index,6);
cont6.nskipt = get_sheet_row_col(input_sheet,start_index,7);
cont6.nskipx = get_sheet_row_col(input_sheet,start_index,8);
cont6.lpopt = get_sheet_row_col(input_sheet,start_index,9);
cont6.icopt = get_sheet_row_col(input_sheet,start_index,10);
cont6.mfopt = get_sheet_row_col(input_sheet,start_index,11);
cont6.nodump = get_sheet_row_col(input_sheet,start_index,12);
cont6.idtail = get_sheet_row_col(input_sheet,start_index,13);
cont6.idtll = get_sheet_row_col(input_sheet,start_index,14);
output_que = char(output_que,'cont6');

%cont.7 given on row 257
start_index = 257;
cont7.tmax = get_sheet_row_col(input_sheet,start_index,1);
cont7.tprint = get_sheet_row_col(input_sheet,start_index,2);
cont7.dumpt = get_sheet_row_col(input_sheet,start_index,3);
cont7.tlplot = get_sheet_row_col(input_sheet,start_index,4);
cont7.tfiche = get_sheet_row_col(input_sheet,start_index,5);
cont7.tpdump = get_sheet_row_col(input_sheet,start_index,6);
output_que = char(output_que,'cont7');

out_que = cellstr(output_que);

%will remember in the input-file generation script, that ENDD and 0 must be at the end of
the
%input file

%================== Store the input data to disk ==================================
```

```
%check to see if the file currently exists -- if it does, delete it:
if exist(strcat(var_file_name,'.mat')) == 2
    delete(strcat(var_file_name,'.mat'));
end
%now save all of the appropriate variables
save(var_file_name,char(out_que(1)));
for i = 2:length(out_que)
   save(var_file_name,char(out_que(i)),'-append');
end
%save the out_que itself so it can be used for re-saving a modified version of the
variables later
save(var_file_name,'out_que','-append');
save(var_file_name,'rods9_pre','-append');

%=========================================================================================
```

Input File Generation

A collection of scripts is used to form a valid VIPRE input file. The input arguments are: a character string representing the desired name of the generated input file – e.g. 'run1', and a character string representing the name of the data file holding the input data generated during the input data extraction phase of the run. – e.g. 'run1_var' or 'run1_var.mat'[j].

The interface for input file generation is not complete in the sense that only sufficient functionality has been implemented as is required for the project at hand. There is more that can be expressed through the input file that has not been encoded here. The vision is that those who require this extra functionality, can make the relatively minor additions to the interface that would be required. It was not deemed practical or desireable to implement every single feature of VIPRE in the script interface.

The code is presented here:

```
%generate_input_file.m
%function generate_input_file(input_file_name,var_file_name) takes as arguements two
strings
%indicating the desired name of the vipre input-file, and the name of the file containing
%the input variables (as generated by get_input_test.m -- or an equivalent function that
%I develop later that does the same thing)


function generate_input_file(input_file_name,var_file_name)

load(var_file_name); %get the variables into the space
%the functions will be called -- hopefully in a good order
fid = fopen(input_file_name,'w');
for i = 1:length(out_que) %one cycle will be required for every entry in the output que
    %every entry in the output que corresponds to a single record in the input file
    %separate functions will be developed to output each individual function to append
    %the desired record to the input file.  The input file will be created only by the
    %output function for vipre.1
    switch char(out_que(i))
    case 'vipre1'
        gen_vipre1(fid,vipre1);
    case 'vipre2'
        gen_vipre2(fid,vipre2);
    case 'geom1'
        gen_geom1(fid,geom1);
    case 'geom2'
        gen_geom2(fid,geom2);
    case 'geom3'
        gen_geom3(fid,geom3);
    case 'geom5'
        %geom5 implies geom6 -- due to the nature of this set of entries,
        %I will generate them together
```

[j] In the case of '*.mat' files, the suffix is optional.

```
    gen_geom5_6(fid,geom5,geom6);
case 'geom7'
    gen_geom7(fid,geom7);
case 'geom11'
    gen_geom11(fid,geom11);
case 'prop1'
    gen_prop1(fid,prop1);
case 'rods1'
    gen_rods1(fid,rods1);
case 'rods2'
    gen_rods2(fid,rods2);
case 'rods3'
    gen_rods3(fid,rods3);
case 'rods5'
    gen_rods5(fid,rods5);
case 'rods9'
    gen_rods9(fid,rods9,rods9_pre);
case 'rods62'
    gen_rods62(fid,rods62);
case 'rods63'
    gen_rods63(fid,rods63);
case 'rods64'
    gen_rods64(fid,rods64);
case 'rods68'
   gen_rods68(fid,rods68);
case 'rods70'
   gen_rods70(fid,rods70);
case 'rods71'
   gen_rods71(fid,rods71);
case 'oper1'
    gen_oper1(fid,oper1);
case 'oper2'
    gen_oper2(fid,oper2);
case 'oper5'
    gen_oper5(fid,oper5);
case 'oper11'
    gen_oper11(fid,oper11);
case 'oper12'
    gen_oper12(fid,oper12);
case 'oper13'
   gen_oper13(fid,oper13);

case 'oper14'
    gen_oper14(fid,oper14);
case 'oper17'
    gen_oper17(fid,oper17);
case 'oper20'
    gen_oper20(fid,oper20);
case 'corr1'
    gen_corr1(fid,corr1);
case 'corr2'
    gen_corr2(fid,corr2);
case 'corr6'
    gen_corr6(fid,corr6);
case 'corr7'
    gen_corr7(fid,corr7);
case 'corr9'
    gen_corr9(fid,corr9);
case 'corr11'
   gen_corr11(fid,corr11);
case 'corr13'
   gen_corr13(fid,corr13);
case 'corr14'
   gen_corr14(fid,corr14);
case 'corr16'
    gen_corr16(fid,corr16);
case 'corr17'
   gen_corr17(fid,corr17);
case 'corr13a'
   gen_corr13a(fid,corr13a);
case 'mixx1'
```

```
            gen_mixx1(fid,mixx1);
        case 'mixx2'
            gen_mixx2(fid,mixx2);
        case 'mixx3'
            gen_mixx3(fid,mixx3);
        case 'drag1'
            gen_drag1(fid,drag1);
        case 'drag2'
            gen_drag2(fid,drag2);
        case 'drag7'
            gen_drag7(fid,drag7);
        case 'drag8'
            gen_drag8(fid,drag8);
        case 'grid1'
            gen_grid1(fid,grid1);
        case 'grid2'
            gen_grid2(fid,grid2);
        case 'grid4'
            gen_grid4(fid,grid4);
        case 'grid6'
            gen_grid6(fid,grid6);
        case 'cont1'
            gen_cont1(fid,cont1);
        case 'cont2'
            gen_cont2(fid,cont2);
        case 'cont3'
            gen_cont3(fid,cont3);
        case 'cont6'
            gen_cont6(fid,cont6);
        case 'cont7'
            gen_cont7(fid,cont7);

        end

end

gen_vipre_ending(fid);
st = fclose(fid);
```

As can be seen, the entries of the character array 'out_que' are traversed from top to bottom. The ordering of the entries in 'out_que' is what enforces the correct ordering of the 'cards' of the input deck as they are generated.

For each entry of 'out_que'-that represents an input card for the VIPRE input deck, a function is called to write that portion of the input deck to the input file. The arguments to each function are an integer representing the file descriptor of the input file, and the Matlab-style data structure that holds the data relevant to that input deck. The code for these functions is presented here.

```
function gen_vipre1(fid,vipre1)

kase = num2str(vipre1.kase);
irstrt = num2str(vipre1.irstrt);
irstep = num2str(vipre1.irstep);
entry = strcat(kase,', ',irstrt,', ',irstep);
count = fprintf(fid,'%s \n',entry);

function gen_vipre2(fid,vipre2)
```

```
count = fprintf(fid,'%s \n',vipre2.text);

function gen_geom1(fid,geom1)

inflag = strcat(geom1.inflag,', ');
nchanl = strcat(num2str(geom1.nchanl),', ');
ncard = strcat(num2str(geom1.ncard),', ');
%ndx = strcat(num2str(geom1.ndx),', ');
ndx = sprintf('%4.0f,',geom1.ndx);
nazone = strcat(num2str(geom1.nazone),', ');
nctyp = strcat(num2str(geom1.nctyp),', ');

if geom1.mbwr == 'empty'
    mbwr = '0';
else
    mbwr = num2str(geom1.mbwr);
end

entry = strcat(inflag,nchanl,ncard,ndx,nazone,nctyp,mbwr);

count = fprintf(fid,'%s \n',entry);

function gen_geom2(fid,geom2)

count = fprintf(fid,'%7.2f, %3.1f,%4.2f \n',geom2.zz,geom2.theta,geom2.sl);

function gen_geom3(fid,geom3)

m = length(geom3);
num_full_records = floor(m/4); %number of records with a full 4 pairs of entries
size_partial_record = mod(m,4); %number of entries in a partial record (if any)

for i = 1:num_full_records
    start_index = 4*(i-1);

    for j = 1:4
        index = start_index + j;
        dx = sprintf('%7.3f',geom3(index).dx);
        xboun = sprintf('%7.3f',geom3(index).xboun);
        if j == 1
            full_entry = strcat(dx,',',xboun,',');
        elseif j == 4
            full_entry = strcat(full_entry,dx,',',xboun);
        else
            full_entry = strcat(full_entry,dx,',',xboun,',');
        end
    end
    %full_entry is formed in full -- send it to the file
    count = fprintf(fid,'%s \n',full_entry);
end


if num_full_records == 0
    start_index = 0;
else
    start_index = start_index + 4;
end

for j = 1:size_partial_record
    index = start_index + j;
    dx = sprintf('%7.3f',geom3(index).dx);
    xboun = sprintf('%7.3f',geom3(index).xboun);
    if j == 1
        partial_entry = strcat(dx,',',xboun,',');
    elseif j==size_partial_record
        partial_entry = strcat(partial_entry,dx,',',xboun);
    else
        partial_entry = strcat(partial_entry,dx,',',xboun,',');
    end
end
```

```
count = fprintf(fid,'%s \n',partial_entry);

function gen_geom5_6(fid,geom5,geom6)

num_channel_types = length(geom5);
[n,m] = size(geom6);

for i = 1:num_channel_types
    mchn = num2str(geom5(i).mchn);
    carea = sprintf('%7.3f',geom5(i).carea);
    cpw = sprintf('%7.3f',geom5(i).cpw);
    cph = sprintf('%7.3f',geom5(i).cph);
    entry = strcat(mchn,',',carea,',',cpw,',',cph);
    count = fprintf(fid,'%s \n',entry);
    entry = '';
    cont = 1; %logical variable to get geom6 values
    j = 1;
    while(cont == 1)
        %there is always at least one channel of the given channel type
        entry = strcat(entry,num2str(geom6(i,j)));
        if j < m
            j = j+1;
            if geom6(i,j) < 1  %no legal channel number is less than one
                cont = 0;
            else
                entry = strcat(entry,',');
            end
        else
            cont = 0;
        end
    end
    count = fprintf(fid,'%s \n',entry); %make the geom6 part of the entry
end

function gen_geom7(fid,geom7)

nk = strcat(num2str(geom7.nk),',');
ngtyp = strcat(num2str(geom7.ngtyp),',');
ngap = strcat(num2str(geom7.ngap),',');
nrow = strcat(num2str(geom7.nrow),',');

if geom7.isym == 'empty'
    isym = ',';
else
    isym = strcat(num2str(geom7.isym),',');
end

if geom7.nsrow == 'empty'
    nsrow = '';
else
    nsrow = num2str(geom7.nsrow);
end

entry = strcat(nk,ngtyp,ngap,nrow,isym,nsrow);

count = fprintf(fid,'%s \n',entry);

function gen_geom11(fid,geom11)

num_entries = length(geom11);

for i = 1:num_entries
    k = strcat(num2str(geom11(i).k),',');
    I = strcat(num2str(geom11(i).I),',');
    J = strcat(num2str(geom11(i).J),',');
    width = sprintf('%7.4f,',geom11(i).width);
    cent = sprintf('%7.4f',geom11(i).cent);
    entry = strcat(k,I,J,width,cent);
    count = fprintf(fid,'%s \n',entry);
end
```

```
function gen_prop1(fid,prop1)

inflag = strcat(prop1.inflag,',');
nprop = strcat(num2str(prop1.nprop),',');
isteam = strcat(num2str(prop1.isteam),',');
nfprop = strcat(num2str(prop1.nfprop),',');
ipvar = num2str(prop1.ipvar);

entry = strcat(inflag,nprop,isteam,nfprop,ipvar);

count = fprintf(fid,'%s \n',entry);

function gen_rods1(fid,rods1)

inflag = 'rods,';

naxp = strcat(num2str(rods1.naxp),',');
nrod = strcat(num2str(rods1.nrod),',');
nc = strcat(num2str(rods1.nc),',');

if rods1.nfuelt ==  'empty'
    nfuelt = '1,';
else
    nfuelt = strcat(num2str(rods1.nfuelt),',');
end

if rods1.nmat == 'empty'
    nmat = '0,';
else
    nmat = strcat(num2str(rods1.nmat),',');
end

if rods1.igpff == 'empty'
    igpff = '0,';
else
    igpff = strcat(num2str(rods1.igpff),',');
end

if rods1.ngpff == 'empty'
    ngpff = '0,';
else
    ngpff = strcat(num2str(rods1.ngpff),',');
end

if rods1.nopt == 'empty'
    nopt = '0,';
else
    nopt = strcat(num2str(rods1.nopt),',');
end

if rods1.ipowv == 'empty'
    ipowv = '0,';
else
    ipowv = strcat(num2str(rods1.nopt),',');
end

if rods1.icpr == 'empty'
    icpr = '0,';
else
    icpr = strcat(num2str(rods1.icpr),',');
end

if rods1.irff == 'empty'
    irff = '0';
else
    irff = num2str(rods1.irff);
end

entry = strcat(inflag,naxp,nrod,nc,nfuelt,nmat,igpff,ngpff,nopt,ipowv,icpr,irff);

count = fprintf(fid,'%s \n',entry);
```

```
function gen_rods2(fid,rods2)

zzh = strcat(num2str(rods2.zzh),',');
zstrt = strcat(num2str(rods2.zstrt),',');

if rods2.nodals == 'empty'
    nodals = '0,';
else
    nodals = strcat(num2str(rods2.nodals),',');
end

if rods2.nodalt == 'empty'
    nodalt = '0';
else
    nodalt = num2str(rods2.nodalt);
end

entry = strcat(zzh,zstrt,nodals,nodalt);

count = fprintf(fid,'%s \n',entry);

function gen_rods3(fid,rods3)

naxn = num2str(rods3.naxn);

count = fprintf(fid,'%s \n',naxn);

function gen_rods5(fid,rods5)

pstar = num2str(rods5.pstar);

count = fprintf(fid,'%s \n',pstar);

function gen_rods9(fid,rods9,rods9_pre)

num_rods = length(rods9_pre);
[n,m] = size(rods9);

for i = 1:num_rods
    I = strcat(num2str(rods9_pre(i).I),',');
    idfuel = strcat(num2str(rods9_pre(i).idfuel),',');
    radial = strcat(num2str(rods9_pre(i).radial),',');
    iaxp =  num2str(rods9_pre(i).iaxp);
    pre_entry = strcat(I,idfuel,radial,iaxp);

    cont = 1;
    j = 1;
    post_entry = '';
    while(cont == 1)
        %will execute at least once and always in pairs of two
        post_entry = strcat(post_entry,',',num2str(rods9(i,j)));
        j = j+1;
        post_entry = strcat(post_entry,',',num2str(rods9(i,j)));

        if j == m %the maximum number of columns in rods9
            cont = 0;
            continue
        end
        %now, look ahead
        j = j+1;
        if rods9(i,j) < 1
            cont = 0;
        end
    end

    entry = strcat(pre_entry,post_entry);
    count = fprintf(fid,'%s \n',entry);
end

end_entry = num2str(0);
```

```
count = fprintf(fid,'%s \n',end_entry);

function gen_rods62(fid,rods62)

I = strcat(num2str(rods62.I),',');
ftype = 'nucl,';
drod = sprintf('%7.4f,',rods62.drod);
dfuel = sprintf('%7.4f,',rods62.dfuel);
nfuel = sprintf('%7.0f,',rods62.nfuel);
dcore = sprintf('%7.4f,',rods62.dcore);
tclad = sprintf('%7.4f',rods62.tclad);

entry = strcat(I,ftype,drod,dfuel,nfuel,dcore,tclad);
count = fprintf(fid,'%s \n',entry);

function gen_rods63(fid,rods63)

iradp = sprintf('%7.0f,',rods63.iradp);
imatf = sprintf('%7.0f,',rods63.imatf);
imatc = sprintf('%7.0f,',rods63.imatc);
igpc = sprintf('%7.0f,',rods63.igpc);
igforc = sprintf('%7.0f,',rods63.igforc);
hgap = sprintf('%7.4f,',rods63.hgap);
ftdens = sprintf('%7.4f,',rods63.ftdens);
fclad = sprintf('%7.4f',rods63.fclad);

entry = strcat(iradp,imatf,imatc,igpc,igforc,hgap,ftdens,fclad);
count = fprintf(fid,'%s \n',entry);

%this function was pirated from the oper11 generating function -- hence the use of the
variable
% 'flofrc' is inexplicable in this context, but works just as well nonetheless
function gen_rods64(fid,rods64)


num_conditions = length(rods64);

full_lines = floor(num_conditions/8);
partial_entry_length = mod(num_conditions,8);

for i = 1:full_lines
    entry = '';
    index = 8*(i-1)+1;
    for j = 1:8

        if j < 8
            flofrc = sprintf('%7.4f,',rods64(index));
            entry = strcat(entry,flofrc);
        else
            flofrc = sprintf('%7.4f',rods64(index));
            entry = strcat(entry,flofrc);
        end
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);
end

index = 8*full_lines + 1;
entry = '';
for j = 1:partial_entry_length

    if j < partial_entry_length
        flofrc = sprintf('%7.4f,',rods64(index));
        entry = strcat(entry,flofrc);
    else
        flofrc = sprintf('%7.4f',rods64(index));
        entry = strcat(entry,flofrc);
    end
    index = index + 1;
end
count = fprintf(fid,'%s \n',entry);
```

```
function gen_rods68(fid,rods68)

num_rods = length(rods68);

for i = 1:num_rods

    I = strcat(num2str(rods68(i).I),',');
    ftype = strcat(num2str(rods68(i).ftype),',');
    drod = sprintf('%7.4f,',rods68(i).drod);
    dfuel = sprintf('%7.4f,',rods68(i).dfuel);
    nfuel = num2str(rods68(i).nfuel);
    entry = strcat(I,ftype,drod,dfuel,nfuel);
    count = fprintf(fid,'%s \n',entry);
end

function gen_rods70(fid,rods70)

n = sprintf('%7.0f,',rods70.n);
nntdp = sprintf('%7.0f,',rods70.nntdp);
rcold = sprintf('%7.4f',rods70.rcold);

entry = strcat(n,nntdp,rcold);
count = fprintf(fid,'%s \n',entry);

function gen_rods71(fid,rods71)

[n,m] = size(rods71);

for i=1:n
   tprop = sprintf('%7.1f', rods71(i,1));
   cpff = sprintf('%7.5f', rods71(i,2));
   thcf = sprintf('%7.5f', rods71(i,3));

   if (rem(i,2)==0)
      entry = strcat(tprop,',',cpff,',',thcf);
      fprintf(fid,'%s\n',entry);
   else
      entry = strcat(tprop,',',cpff,',',thcf,',');
      fprintf(fid,entry);
   end
end

function gen_oper1(fid,oper1)

inflag = 'oper,';
ih = strcat(num2str(oper1.ih),',');
ig = strcat(num2str(oper1.ig),',');
isp = strcat(num2str(oper1.isp),',');
npowr = strcat(num2str(oper1.npowr),',');
ndnb = strcat(num2str(oper1.ndnb),',');

if oper1.irun == 'empty'
    irun = '1,';
else
    irun = strcat(num2str(oper1.irun),',');
end

if oper1.ifcvr == 'empty'
    ifcvr = '0,';
else
    ifcvr = strcat(num2str(oper1.ifcvr),',');
end

if oper1.luf == 'empty'
    luf = '0,';
else
    luf = strcat(num2str(oper1.luf),',');
end

if oper1.ihbal == 'empty'
```

```
    ihbal = '0';
else
    ihbal = num2str(oper1.ihbal);
end

entry = strcat(inflag,ih,ig,isp,npowr,ndnb,irun,ifcvr,luf,ihbal);

count = fprintf(fid,'%s \n',entry);

function gen_oper2(fid,oper2)

dps = sprintf('%7.4f,',oper2.dps);
dnbrl = sprintf('%7.4f,',oper2.dnbrl);
fcool = sprintf('%7.4f,',oper2.fcool);
dnbrc = sprintf('%7.4f,',oper2.dnbrc);

if oper2.ihrod == 'empty'
    ihrod = '0';
else
    ihrod = num2str(oper2.ihrod);
end

entry = strcat(dps,dnbrl,fcool,dnbrc,ihrod);

count = fprintf(fid,'%s \n',entry);

function gen_oper5(fid,oper5)

pref = sprintf('%7.4f,',oper5.pref);
hin = sprintf('%7.4f,',oper5.hin);
gin = sprintf('%7.4f,',oper5.gin);
pwrinp = sprintf('%7.4f,',oper5.pwrinp);

if oper5.hout == 'empty'
    hout = '0';
else
    hout = num2str(oper5.hout);
end

entry = strcat(pref,hin,gin,pwrinp,hout);

count = fprintf(fid,'%s \n',entry);

function gen_oper11(fid,oper11)


num_conditions = length(oper11.flofrc);

full_lines = floor(num_conditions/8);
partial_entry_length = mod(num_conditions,8);

for i = 1:full_lines
    entry = '';
    index = 8*(i-1)+1;
    for j = 1:8

        if j < 8
            flofrc = sprintf('%7.4f,',oper11.flofrc(index));
            entry = strcat(entry,flofrc);
        else
            flofrc = sprintf('%7.4f',oper11.flofrc(index));
            entry = strcat(entry,flofrc);
        end
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);
end

index = 8*full_lines + 1;
entry = '';
for j = 1:partial_entry_length
```

```
    if j < partial_entry_length
        flofrc = sprintf('%7.4f,',oper11.flofrc(index));
        entry = strcat(entry,flofrc);
    else
        flofrc = sprintf('%7.4f',oper11.flofrc(index));
        entry = strcat(entry,flofrc);
    end
    index = index + 1;
end
count = fprintf(fid,'%s \n',entry);

function gen_oper12(fid,oper12)

if oper12.np == 'empty'
    np = '0,';
else
    np = strcat(num2str(oper12.np),',');
end

if oper12.nh == 'empty'
    nh = '0,';
else
    nh = strcat(num2str(oper12.nh),',');
end

if oper12.ng == 'empty'
    ng = '0,';
else
    ng = strcat(num2str(oper12.ng),',');
end

if oper12.nq == 'empty'
    nq = '0,';
else
    nq = strcat(num2str(oper12.nq),',');
end

if oper12.nnath == 'empty'
    nnath = '0,';
else
    nnath = strcat(num2str(oper12.nnath),',');
end

if oper12.nnatg == 'empty'
    nnatg = '0';
else
    nnatg = num2str(oper12.nnatg);
end
entry = strcat(np,nh,ng,nq,nnath,nnatg);

count = fprintf(fid,'%s \n',entry);

function gen_oper13(fid,oper13)

num_conditions = length(oper13);

full_lines = floor(num_conditions/8);
partial_entry_length = mod(num_conditions,8);

for i = 1:full_lines
    entry = '';
    index = 8*(i-1)+1;
    for j = 1:8

        if j < 8
            flofrc = sprintf('%7.4f,',oper13(index));
            entry = strcat(entry,flofrc);
        else
            flofrc = sprintf('%7.4f',oper13(index));
            entry = strcat(entry,flofrc);
```

```
        end
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);
end

index = 8*full_lines + 1;
entry = '';
for j = 1:partial_entry_length

    if j < partial_entry_length
        flofrc = sprintf('%7.4f,',oper13(index));
        entry = strcat(entry,flofrc);
    else
        flofrc = sprintf('%7.4f',oper13(index));
        entry = strcat(entry,flofrc);
    end
    index = index + 1;
end

if(partial_entry_length >= 1)
    count = fprintf(fid,'%s \n',entry);
end

function gen_oper14(fid,oper14)

num_conditions = length(oper14);

full_lines = floor(num_conditions/8);
partial_entry_length = mod(num_conditions,8);

for i = 1:full_lines
    entry = '';
    index = 8*(i-1)+1;
    for j = 1:8

        if j < 8
            flofrc = sprintf('%7.4f,',oper14(index));
            entry = strcat(entry,flofrc);
        else
            flofrc = sprintf('%7.4f',oper14(index));
            entry = strcat(entry,flofrc);
        end
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);
end

index = 8*full_lines + 1;
entry = '';
for j = 1:partial_entry_length

    if j < partial_entry_length
        flofrc = sprintf('%7.4f,',oper14(index));
        entry = strcat(entry,flofrc);
    else
        flofrc = sprintf('%7.4f',oper14(index));
        entry = strcat(entry,flofrc);
    end
    index = index + 1;
end

if(partial_entry_length >= 1)
    count = fprintf(fid,'%s \n',entry);
end

function gen_oper17(fid,oper17)

num_conditions = length(oper17);

full_lines = floor(num_conditions/8);
```

```
partial_entry_length = mod(num_conditions,8);

for i = 1:full_lines
    entry = '';
    index = 8*(i-1)+1;
    for j = 1:8

        if j < 8
            flofrc = sprintf('%7.4f,',oper17(index));
            entry = strcat(entry,flofrc);
        else
            flofrc = sprintf('%7.4f',oper17(index));
            entry = strcat(entry,flofrc);
        end
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);
end

index = 8*full_lines + 1;
entry = '';
for j = 1:partial_entry_length

    if j < partial_entry_length
        flofrc = sprintf('%7.4f,',oper17(index));
        entry = strcat(entry,flofrc);
    else
        flofrc = sprintf('%7.4f',oper17(index));
        entry = strcat(entry,flofrc);
    end
    index = index + 1;
end

if(partial_entry_length >= 1)
    count = fprintf(fid,'%s \n',entry);
end

function gen_oper20(fid,oper20)

num_conditions = length(oper20);

full_lines = floor(num_conditions/8);
partial_entry_length = mod(num_conditions,8);

for i = 1:full_lines
    entry = '';
    index = 8*(i-1)+1;
    for j = 1:8

        if j < 8
            flofrc = sprintf('%7.4f,',oper20(index));
            entry = strcat(entry,flofrc);
        else
            flofrc = sprintf('%7.4f',oper20(index));
            entry = strcat(entry,flofrc);
        end
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);
end

index = 8*full_lines + 1;
entry = '';
for j = 1:partial_entry_length

    if j < partial_entry_length
        flofrc = sprintf('%7.4f,',oper20(index));
        entry = strcat(entry,flofrc);
    else
        flofrc = sprintf('%7.4f',oper20(index));
        entry = strcat(entry,flofrc);
```

```
    end
    index = index + 1;
end

if(partial_entry_length >= 1)
    count = fprintf(fid,'%s \n',entry);
end

function gen_corr1(fid,corr1)

inflag = 'corr,';
ncor = strcat(num2str(corr1.ncor),',');
nhtc = strcat(num2str(corr1.nhtc),',');
if corr1.ixchf == 'empty'
    ixchf = '0';
else
    ixchf = num2str(corr1.ixchf);
end


entry = strcat(inflag,ncor,nhtc,ixchf);

count = fprintf(fid,'%s \n',entry);

function gen_corr2(fid,corr2)

    nscvd = strcat(corr2.nscvd,',');
    nblvd = strcat(corr2.nblvd,',');
    nfrml = strcat(corr2.nfrml,',');
    nhtwl = strcat(corr2.nhtwl,',');
     entry = strcat(nscvd,nblvd,nfrml,nhtwl);
     count = fprintf(fid,'%s \n',entry);

function gen_corr6(fid,corr6)

if strcmp(corr6.nfcon,'empty') == 1
    nfcon = 'epri,';
else
    nfcon = strcat(corr6.nfcon,',');
end

if strcmp(corr6.nsubc,'empty') == 1
    nsubc = 'thsp,';
else
    nsubc = strcat(corr6.nsubc,',');
end

if strcmp(corr6.nsatb,'empty') == 1
    nsatb = 'thsp,';
else
    nsatb = strcat(corr6.nsatb,',');
end

if strcmp(corr6.nchfc,'empty') == 1
    nchfc = 'epri,';
else
    nchfc = strcat(corr6.nchfc,',');
end

if strcmp(corr6.ntrnb,'empty') == 1
    ntrnb = 'cond,';
else
    ntrnb = strcat(corr6.ntrnb,',');
end

if strcmp(corr6.nflmb,'empty') == 1
    nflmb = 'g5.7,';
else
    nflmb = strcat(corr6.nflmb,',');
end
```

```
entry = strcat(nfcon,nsubc,nsatb,nchfc,ntrnb,nflmb);

count = fprintf(fid,'%s \n',entry);

function gen_corr7(fid,corr7)

cdb = sprintf('%7.4f',corr7.cdb);

count = fprintf(fid,'%s \n',cdb);

function gen_corr7(fid,corr7)

cdb = sprintf('%7.4f',corr7.cdb);

count = fprintf(fid,'%s \n',cdb);

function gen_corr9(fid,corr9)

[num_corrs,m] = size(corr9.nchf);

entry = '';
for i = 1:num_corrs
    entry = strcat(entry,corr9.nchf(i,:),',');
end

count = fprintf(fid,'%s \n',entry);

function gen_corr11(fid,corr11)

tdcl = sprintf('%7.4f,',corr11.tdcl);
spk = sprintf('%7.4f,',corr11.spk);
flgrd = sprintf('%7.4f',corr11.flgrd);

entry = strcat(tdcl,spk,flgrd);

count = fprintf(fid,'%s \n',entry);

function gen_corr13(fid,corr13)

fvane= sprintf('%7.4f',corr13.fvane);

entry = fvane;
count = fprintf(fid,'%s \n',entry);

function gen_corr14(fid,corr14)
%this is a hack -- only works for models where all of the channels are of a given type.
nchan = strcat(num2str(corr14.nchan),',');
mtp = num2str(corr14.mtp);
entry = strcat(nchan,mtp);
count = fprintf(fid,'%s \n',entry);

function gen_corr16(fid,corr16)

kbwr = strcat(num2str(corr16.kbwr),',');
nuc = strcat(num2str(corr16.nuc),',');
cgrid = sprintf('%7.4f',corr16.cgrid);

entry = strcat(kbwr,nuc,cgrid);

count = fprintf(fid,'%s \n',entry);

function gen_corr17(fid,corr17)

mac = num2str(corr17.mac);

count = fprintf(fid,'%s \n',mac);
```

For using the KfK correlation, added to VIPRE by Jacopo Saccherri,[10] the additional card, entitled "corr13a" is required. According to the input rules for VIPRE this card must be placed after the supplementary input cards for the other correlation. For this reason, it is not in it's logical location (e.g. just after the entry for corr13).

```
function gen_corr13a(fid,corr13a)

h= sprintf('%7.4f,',corr13a.h);
p= sprintf('%7.4f,',corr13a.p);
d= sprintf('%7.4f',corr13a.d);

entry = strcat(h,p,d);

count = fprintf(fid,'%s \n',entry);

function gen_mixx1(fid,mixx1)

nflag = 'mixx,';

nscbc = strcat(num2str(mixx1.nscbc),',');
nbbc = strcat(num2str(mixx1.nbbc),',');
mixk = num2str(mixx1.mixk);
entry = strcat(nflag,nscbc,nbbc,mixk);
count = fprintf(fid,'%s \n',entry);

function gen_mixx2(fid,mixx2)

ftm = sprintf('%7.4f,',mixx2.ftm);
abeta = sprintf('%7.4f,',mixx2.abeta);
bbeta = sprintf('%7.4f',mixx2.bbeta);
entry = strcat(ftm,abeta,bbeta);
count = fprintf(fid,'%s \n',entry);

function gen_mixx3(fid,mixx3)

num_entries = length(mixx3);

num_full_entries = floor(num_entries/8);
size_partial_entry = mod(num_entries,8);

for i = 1:num_full_entries
    entry = '';
    index = 8*(i-1) + 1;
    for j = 1:8
        num = sprintf('%7.4f,',mixx3(index));
        entry = strcat(entry,num);
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);
end

index = 8*num_full_entries + 1;
entry = '';
for i = 1:size_partial_entry
    num = sprintf('%7.4f',mixx3(index));
    if i == size_partial_entry
        entry = strcat(entry,num);
    else
        entry = strcat(entry,num,',');
    end
    index = index + 1;
end
count = fprintf(fid,'%s \n',entry);

function gen_drag1(fid,drag1)
```

```
inflag = 'drag,';
nchtp = strcat(num2str(drag1.nchtp),',');
ngptp = strcat(num2str(drag1.ngptp),',');
%kijopt = strcat(num2str(drag1.kijopt),',');
kijopt = num2str(drag1.kijopt);
entry = strcat(inflag,nchtp,ngptp,kijopt);
count = fprintf(fid,'%s \n',entry);

function gen_drag2(fid,drag2)

atf = sprintf('%7.4f,',drag2.atf);
btf = sprintf('%7.4f,',drag2.btf);
ctf = sprintf('%7.4f,',drag2.ctf);
alf = sprintf('%7.4f,',drag2.alf);
blf = sprintf('%7.4f,',drag2.blf);
clf = sprintf('%7.4f',drag2.clf);
entry = strcat(atf,btf,ctf,alf,blf,clf);
count = fprintf(fid,'%s \n',entry);

function gen_drag7(fid,drag7)

ddok = sprintf('%7.4f,',drag7.ddok);
ppitch = sprintf('%7.4f',drag7.ppitch);
entry = strcat(ddok,ppitch);
count = fprintf(fid,'%s \n',entry);

function gen_drag8(fid,drag8)

atg = sprintf('%7.4f,',drag8.atg);
btg = sprintf('%7.4f,',drag8.btg);
ctg = sprintf('%7.4f,',drag8.ctg);
alg = sprintf('%7.4f,',drag8.alg);
blg = sprintf('%7.4f,',drag8.blg);
clg = sprintf('%7.4f',drag8.clg);
entry = strcat(atg,btg,ctg,alg,blg,clg);
count = fprintf(fid,'%s \n',entry);

function gen_grid1(fid,grid1)

inflag = 'grid,';
kopt = strcat(num2str(grid1.kopt),',');
nkcor = num2str(grid1.nkcor);
entry = strcat(inflag,kopt,nkcor);
count = fprintf(fid,'%s \n',entry);

function gen_grid2(fid,grid2)

num_coeff = length(grid2);
full_records = floor(num_coeff/8);
length_partial_record = mod(num_coeff,8);

for i = 1:full_records
    entry = '';
    index = 8*(i-1)+1;
    for j = 1:8
        num = sprintf('%7.4f,',grid2(index));
        entry = strcat(entry,num);
        index = index + 1;
    end

    count = fprintf(fid,'%s \n',entry);
end

index = 8*full_records + 1;
entry = '';
for j = 1:length_partial_record
    num = sprintf('%7.4f',grid2(index));
    if j == length_partial_record
        entry = strcat(entry,num);
    else
        entry = strcat(entry,num,',');
```

```
    end
    index = index + 1;
end
count = fprintf(fid,'%s \n',entry);

function gen_grid4(fid,grid4);

nci = strcat(num2str(grid4.nci),',');
nlev = num2str(grid4.nlev);
entry = strcat(nci,nlev);
count = fprintf(fid,'%s \n',entry);

function gen_grid6(fid,grid6)

num_entries = length(grid6);

num_pairs = num_entries/2;

num_records = ceil(num_pairs/8);

num_full_records = floor(num_pairs/8);
len_partial_record = mod(num_pairs,8);
if len_partial_record > 4
    rows_partial_record = 2;
else
    rows_partial_record = 1;
end


for i = 1:num_full_records
    index = 16*(i-1)+1;
   for rows = 1:2
       entry = '';
        for j = 1:4
         axj = sprintf('%7.4f,',grid6(index));
         index = index + 1;
         if (j == 4) & (rows == 2)
             kor = num2str(grid6(index));
         else
            kor = strcat(num2str(grid6(index)),',');
        end
         entry = strcat(entry,axj,kor);
         index = index + 1;
        end
        if rows == 1
         entry = strcat(entry,'?');
        end
        count = fprintf(fid,'%s \n',entry);
   end
end

index = 16*num_full_records + 1;

%must account for the possibility that a single record will now
%require more than one row of output. (i.e. need a continuing '?')

if rows_partial_record == 1
    num_entries_remaining = num_entries - (index - 1);
    entry = '';
    for j = 1:2:num_entries_remaining
        axj = sprintf('%7.4f,',grid6(index));
        index = index+1;
        if j == num_entries_remaining
            kor = num2str(grid6(index));
        else
            kor = strcat(num2str(grid6(index)),',');
        end
        entry = strcat(entry,axj,kor);
        index = index+1;
    end
    count = fprintf(fid,'%s \n',entry);
```

```
elseif rows_partial_record == 2
    %do the first row
    entry = '';
    for j = 1:4
        axj = sprintf('%7.4f,',grid6(index));
        index = index + 1;
        kor = strcat(num2str(grid6(index)),',');
        entry = strcat(entry,axj,kor);
        index = index + 1;
    end
    entry = strcat(entry,'?');
    count = fprintf(fid,'%s \n',entry);
    %now do the remaining full/partial row
    num_entries_remaining = num_entries - (index - 1);
    entry = '';
    for j = 1:2:num_entries_remaining
        axj = sprintf('%7.4f,',grid6(index));
        index = index + 1;
        if j == num_entries_remaining
            kor = num2str(grid6(index));
        else
            kor = strcat(num2str(grid6(index)),',');
        end
        entry = strcat(entry,axj,kor);
        index = index + 1;
    end
    count = fprintf(fid,'%s \n',entry);

end

%generate the terminating 0 entry for grid.4
count = fprintf(fid,'0 \n');

function gen_cont1(fid,cont1)

inflag = num2str(cont1.inflag);
count = fprintf(fid,'%s \n',inflag);

function gen_cont2(fid,cont2)

if strcmp(cont2.ttdum,'empty') == 1
    ttdum = '0.0,';
else
    ttdum = sprintf('%7.4f,',cont2.ttdum);
end

if strcmp(cont2.ndtdum,'empty') == 1
    ndtdum = '0,';
else
    ndtdum = strcat(num2str(cont2.ndtdum),',');
end

if strcmp(cont2.ntrydm,'empty') == 1
    ntrydm = '20,';
else
    ntrydm = strcat(num2str(cont2.ntrydm),',');
end

if strcmp(cont2.itry,'empty') == 1
    itry ='50,';
else
    itry = strcat(num2str(cont2.itry),',');
end

if strcmp(cont2.itrym,'empty') == 1
    itrym = '2,';
else
    itrym = strcat(num2str(cont2.itrym),',');
end
```

```
if strcmp(cont2.idrect,'empty') == 1
    idrect = '1,';
else
    idrect = strcat(num2str(cont2.idrect),',');
end

if strcmp(cont2.itstep,'empty') == 1
    itstep = '0,';
else
    itstep = strcat(num2str(cont2.itstep),',');
end

if strcmp(cont2.itmod,'empty') == 1
    itmod = '1';
else
    itmod = num2str(cont2.itmod);
end

entry = strcat(ttdum,ndtdum,ntrydm,itry,itrym,idrect,itstep,itmod);
count = fprintf(fid,'%s \n',entry);

function gen_cont3(fid,cont3)

if strcmp(cont3.werrx,'empty') == 1
    werrx = '0.10,';
else
    werrx = sprintf('%7.4f,',cont3.werrx);
end

if strcmp(cont3.werry,'empty') == 1
    werry = '0.00001,';
else
    werry = sprintf('%7.5f,',cont3.werry);
end

if strcmp(cont3.ferror,'empty') == 1
    ferror = '0.001,';
else
    ferror = sprintf('%7.5f,',cont3.ferror);
end

if strcmp(cont3.terror,'empty') == 1
    terror = '0.05,';
else
    terror = sprintf('%7.5f,',cont3.terror);
end

if strcmp(cont3.htcerr,'empty') == 1
    htcerr = '0.01,';
else
    htcerr = sprintf('%7.5f,',cont3.htcerr);
end

if strcmp(cont3.dampng,'empty') == 1
    dampng = '0.9,';
else
    dampng = sprintf('%7.5f,',cont3.dampng);
end

if strcmp(cont3.accely,'empty') == 1
    accely = '1.5,';
else
    accely = sprintf('%7.4f,',cont3.accely);
end

if strcmp(cont3.accelf,'empty') == 1
    accelf = '1.0';
else
    accelf = sprintf('%7.4f',cont3.accelf);
end
```

```
entry = strcat(werrx,werry,ferror,terror,htcerr,dampng,accely,accelf);
count = fprintf(fid,'%s \n',entry);

function gen_cont6(fid,cont6)

nout = strcat(num2str(cont6.nout),',');
npchan = strcat(num2str(cont6.npchan),',');
npgap = strcat(num2str(cont6.npgap),',');
nprod = strcat(num2str(cont6.nprod),',');
npchf = strcat(num2str(cont6.npchf),',');

if strcmp(cont6.npwl,'empty') == 1
    npwl = '0,';
else
    npwl = strcat(num2str(cont6.npwl),',');
end

if strcmp(cont6.nskipt,'empty') == 1
    nskipt = '1,';
else
    nskipt = strcat(num2str(cont6.nskipt),',');
end

if strcmp(cont6.nskipx,'empty') == 1
    nskipx = '1,';
else
    nskipx = strcat(num2str(cont6.nskipx),',');
end

if strcmp(cont6.lpopt,'empty') == 1
    lpopt = '0,';
else
    lpopt = strcat(num2str(cont6.lpopt),',');
end

if strcmp(cont6.icopt,'empty') == 1
    icopt = '0,';
else
    icopt = strcat(num2str(cont6.icopt),',');
end

if strcmp(cont6.mfopt,'empty') == 1
    mfopt = '0,';
else
    mfopt = strcat(num2str(cont6.mfopt),',');
end

if strcmp(cont6.nodump,'empty') == 1
    nodump = '0,';
else
    nodump = strcat(num2str(cont6.nodump),',');
end

if strcmp(cont6.idtail,'empty') == 1
    idtail = '0,';
else
    idtail = strcat(num2str(cont6.idtail),',');
end

if strcmp(cont6.idtl1,'empty') == 1
    idtl1 = '0';
else
    idtl1 = num2str(cont6.idtl1);
end

entry =
strcat(nout,npchan,npgap,nprod,npchf,npwl,nskipt,nskipx,lpopt,icopt,mfopt,nodump,idtail,i
dtl1);
count = fprintf(fid,'%s \n',entry);

function gen_cont7(fid,cont7)
```

```
if strcmp(cont7.tmax,'empty') == 1
    tmax = '100.0,';
else
    tmax = sprintf('%7.4f,',cont7.tmax);
end

if strcmp(cont7.tprint,'empty') == 1
    tprint = ',';
else
    tprint = sprintf('%7.4f,',cont7.tprint);
end

if strcmp(cont7.dumpt,'empty') == 1
    dumpt = ',';
else
    dumpt = sprintf('%7.4f,',cont7.dumpt);
end

if strcmp(cont7.tlplot,'empty') == 1
    tlplot = ',';
else
    tlplot = sprintf('%7.4f,',cont7.tlplot);
end

if strcmp(cont7.tfiche,'empty') == 1
    tfiche = ',';
else
    tfiche = sprintf('%7.4f,',cont7.tfiche);
end

if strcmp(cont7.tpdump,'empty') ==1
    tpdump = '';
else
    tpdump = sprintf('%7.4f',cont7.tpdump);
end

entry = strcat(tmax,tprint,dumpt,tlplot,tfiche,tpdump);
count = fprintf(fid,'%s \n',entry);

function gen_vipre_ending(fid)

count = fprintf(fid,'endd \n');
count = fprintf(fid,'0');
```

Vipre Execution Coordination

Once the input data is extracted and encoded into a valid input file, it is time to start a vipre run. The following function performs this task. The name, 'vipre_run_jon.m' is indicitive of the version of VIPRE that was in use – a version we called 'jon_vipre'.

The basic structure is to copy the input file to the directory where the VIPRE executable resides. The mex-function 'call_jon_vipre' causes VIPRE to be initiated with the given input file. The resultant files are copied to the original working directory where they are processed by the 'parse_output_sv4_trans.m' script. The VIPRE directory is cleaned of the used files.

```
function vipre_run_jon(inputfile,var_file)

    outdirectory = pwd;
    current_dir = pwd;
    copyfile(inputfile,'c:\matlab_sv13\bin\vipre\input.txt');
    %transfer control to the vipre directory
    cd('c:\matlab_sv13\bin\vipre');
    %ensure that I have access to the callVipre.dll
    addpath('c:\matlab_sv13\bin\vipre');
    %execute the system call
    call_jon_vipre;
    %now delete the files
    % 'Vipre completed, collecting data'
    if exist('restout.txt')
       delete('restout.txt');
    end

    delete('input.txt');
    delete('calcmp');
    delete('fiche');
    delete('rasp');
    delete('scrac');
    %copy the useful files
    copyfile('chfdat.txt',strcat(outdirectory,'\','chfdat.txt'));
    copyfile('chsum.txt',strcat(outdirectory,'\','chsum.txt'));
    copyfile('celldat.txt',strcat(outdirectory,'\','celldat.txt'));
    copyfile('hotchnl.txt',strcat(outdirectory,'\','hotchnl.txt'));
    copyfile('fuelthr.txt',strcat(outdirectory,'\','fuelthr.txt'));
    %delete the left-over files
    delete('chsum.txt');
    delete('chfdat.txt');
    delete('outptt.txt');
    delete('chfdmp.txt');
    delete('celldat.txt');
    delete('hotchnl.txt');
    delete('fuelthr.txt');
    delete('restout');
    cd(current_dir);
    parse_output_sv4_trans;
    %'Data Collected and Parsed'
```

Output Data Parsing

As shown in the previous section, once VIPRE is through running, the data files are copied to the Matlab working directory. Here they are parsed. The data files generated by the VIPRE executable that are intended for further processing by the Matlab-VIPRE Interface are simply ASCII-formatted files filled with space-delimited numbers. The structure of these numbers is known based on the modifications made to the VIPRE source-code. The output data parsing functions exist to arrange this output data into Matlab-type arrays that are easy for the programmer to recognize and access.

This particular function is, admittedly rather fragile in that it must be modified manually in the event that there are expected changes to the structure of the output data. Examples of this situation include: Changes to the number of rods, channels, gaps, transient time steps, number of CHF correlations to be used, and number of axial zones to skip between data outputs.

The source code is provided here:

```
%parse_output_sv4_trans.m

%some data needed to parse the output files:
NUM_TIME_STEPS = 272;
NUM_CHANNELS = 20;
NUM_AXIAL_NODES = 97;
%NUM_CHF_CORR is eliminated for transient calculations.  There will be the option for
only one.
%NUM_CHF_CORR = 1;
NUM_RODS = 24;
%CHF_DATA_COLUMNS = 12;
%CH_SUM_DATA_COLUMS = 7; %will leave out the first column -- the channel numbers since
they are
%numbered consecutively
NUM_AXIAL_DATA = ceil(NUM_AXIAL_NODES/2); %based on one observation -- may need revision

load chfdat.txt;
load chsum.txt;
load celldat.txt;
load hotchnl.txt;
%load fuelthr.txt;<- only used when there is a fuel model - not used for dummy rods
%process this file first
axial_zone_bottom = chfdat(1:NUM_AXIAL_DATA,1); %this is ok for the transient case -
numbers
axial_zone_top = chfdat(1:NUM_AXIAL_DATA,1); %don't change
%only need once, since it is the same for every channel
%============================initialize data ====================
channel_dnbr = zeros(NUM_TIME_STEPS,NUM_AXIAL_DATA,NUM_CHANNELS);
channel_rod_number = zeros(NUM_TIME_STEPS,NUM_AXIAL_DATA,NUM_CHANNELS);
channel_surface_hf = channel_dnbr;
channel_chf = channel_dnbr;
channel_uniform_chf = channel_dnbr;
channel_axial_flux_factor = channel_dnbr;
channel_grid_factor = channel_dnbr;
channel_unheated_wall_factor = channel_dnbr;
channel_mass_velocity = channel_dnbr;
```

```
channel_temp = zeros(NUM_CHANNELS,1);
channel_enthalpy = channel_temp;
channel_density = channel_temp;
channel_quality = channel_temp;
channel_void_fraction = channel_temp;
channel_flow_rate = channel_temp;
channel_mass_flux = channel_temp;
%================================================================
%========================== initialize new data ================
channel_distance = zeros(NUM_TIME_STEPS,NUM_AXIAL_DATA+1,NUM_CHANNELS);
channel_delta_p_local = channel_distance;
channel_flowrate_local = channel_distance;
channel_velocity_local = channel_distance;
channel_mass_flux_local_local = channel_distance;
%skip one row for channel distance -- this shouldn't have been saved... but
%oh well....
channel_density_local = channel_distance;
channel_enthalpy_local = channel_distance;
channel_void_fraction_local = channel_distance;
channel_quality_local = channel_distance;
channel_temp_local = channel_distance;
channel_linear_heat_rate_local = channel_distance;
%================================================================
%====================== hot channel data ========================
corr_mdnbr = zeros(NUM_TIME_STEPS,1);
corr_hot_channel = corr_mdnbr;
corr_hot_rod = corr_mdnbr;
corr_mdnbr_axial_level = corr_mdnbr;
corr_mdnbr_eq_quality = corr_mdnbr;
corr_mdnbr_heat_flux = corr_mdnbr;
corr_mdnbr_chf = corr_mdnbr;
transient_time = corr_mdnbr;
%================================================================
%======================== Fuel Thermal Data =====================
rod_hx_coeff = zeros(NUM_TIME_STEPS,NUM_AXIAL_DATA,NUM_RODS);
rod_clad_temp_out = rod_hx_coeff;
rod_clad_temp_ave = rod_hx_coeff;
rod_clad_temp_in  = rod_hx_coeff;
rod_gap_cond = rod_hx_coeff;
rod_fuel_temp_out = rod_hx_coeff;
rod_fuel_temp_ave = rod_hx_coeff;
rod_fuel_temp_cl = rod_hx_coeff;
%================================================================

index_start = 1;

for ch = 1:NUM_TIME_STEPS
    corr_mdnbr(ch) = hotchnl(ch,6);
    corr_hot_channel(ch) = hotchnl(ch,7);
    corr_hot_rod(ch) = hotchnl(ch,8);
    corr_mdnbr_axial_level(ch) = hotchnl(ch,9);
    corr_mdnbr_eq_quality(ch) = hotchnl(ch,11);
    corr_mdnbr_heat_flux(ch) = hotchnl(ch,12);
    corr_mdnbr_chf(ch) = hotchnl(ch,13);
    transient_time(ch) = hotchnl(ch,14);
    for j = 1:NUM_CHANNELS
        index_end = index_start + NUM_AXIAL_DATA-1;
        channel_dnbr(ch,:,j) = chfdat(index_start:index_end,3);
        channel_rod_number(ch,:,j) = chfdat(index_start:index_end,4);
        channel_surface_hf(ch,:,j) = chfdat(index_start:index_end,5);
        channel_chf(ch,:,j) = chfdat(index_start:index_end,6);
        channel_uniform_chf(ch,:,j) = chfdat(index_start:index_end,7);
        channel_axial_flux_factor(ch,:,j) = chfdat(index_start:index_end,8);
        channel_grid_factor(ch,:,j) = chfdat(index_start:index_end,9);
        channel_unheated_wall_factor(ch,:,j) = chfdat(index_start:index_end,10);
        channel_mass_velocity(ch,:,j) = chfdat(index_start:index_end,11);
        channel_equilibrium_quality(ch,:,j) = chfdat(index_start:index_end,12);
        index_start = index_start + NUM_AXIAL_DATA; %increment down to the next set of
channel data
    end
```

```
end

index_start = 1;
for ts = 1:NUM_TIME_STEPS
    for j = 1:NUM_CHANNELS
        index_end = index_start + NUM_AXIAL_DATA; %allows for one extra row compared to
above code
        channel_distance(ts,:,j) = celldat(index_start:index_end,1);
        channel_delta_p_local(ts,:,j) = celldat(index_start:index_end,2);
        channel_flowrate_local(ts,:,j) = celldat(index_start:index_end,3);
        channel_velocity_local(ts,:,j) = celldat(index_start:index_end,4);
        channel_mass_flux_local_local(ts,:,j) = celldat(index_start:index_end,5);
        %skip one row for channel distance -- this shouldn't have been saved... but
       %oh well....
        channel_density_local(ts,:,j) = celldat(index_start:index_end,7);
        channel_enthalpy_local(ts,:,j) = celldat(index_start:index_end,8);
        channel_void_fraction_local(ts,:,j) = celldat(index_start:index_end,9);
        channel_quality_local(ts,:,j) = celldat(index_start:index_end,10);
        channel_temp_local(ts,:,j) = celldat(index_start:index_end,11);
        channel_linear_heat_rate_local(ts,:,j) = celldat(index_start:index_end,12);
        index_start = index_end + 1;
    end
end

This section is commented-out because it is only used for runs where there exists a fuel
rod model.  For 'dummy' rods, this data is not generated or collected.
%index_start = 1;
%for ts = 1:NUM_TIME_STEPS
%    for j = 1:NUM_RODS
%        index_end = index_start + NUM_AXIAL_DATA-1;
%        rod_hx_coeff(ts,:,j) = fuelthr(index_start:index_end,1);
%        rod_clad_temp_out(ts,:,j) = fuelthr(index_start:index_end,2);
%        rod_clad_temp_ave(ts,:,j) = fuelthr(index_start:index_end,3);
%        rod_clad_temp_in(ts,:,j)  = fuelthr(index_start:index_end,4);
%        rod_gap_cond(ts,:,j) = fuelthr(index_start:index_end,5);
%        rod_fuel_temp_out(ts,:,j) = fuelthr(index_start:index_end,6);
%        rod_fuel_temp_ave(ts,:,j) = fuelthr(index_start:index_end,7);
%        rod_fuel_temp_cl(ts,:,j) = fuelthr(index_start:index_end,8);
%        index_start = index_end + 1;
%    end
%end

channel_enthalpy = chsum(:,2);
channel_temp = chsum(:,3);
channel_density = chsum(:,4);
channel_quality = chsum(:,5);
channel_void_fraction = chsum(:,6);
channel_flow_rate = chsum(:,7);
channel_mass_flux = chsum(:,8);

clear chfdat;
clear chsum;
clear celldat;
clear hotchnl;
%clear fuelthr;
save channel_data; %save the remaining data to a file so that it may be re-loaded later
```

Modifications to VIPRE

Several modifications to the VIPRE source code were required in order to allow for the script-interface.[k] Some of the changes were merely for convenience – such as specifying that the input file should be named "input.txt" instead of simply "input".[l] Other changes were absolutely essential – re-dimensioning the data arrays to allow for more than 10 time-steps.

Additional Output Files

In order to allow for the scripted interface, it was desirable to obtain the normal program output data in a form more easily processed by the script interface. In particular, it is inconvenient (at least for a Matlab-based script interface) to have text interspersed with numeric data. There are no simple, high-level Matlab commands that deal effectively with the mixture of text and numerics in a file. On the other hand, it is a trivial task to load a file into the Matlab workspace that is space or tab-delimited numbers[m].

The most important principle guiding the modifications to the VIPRE source code was: DO NOT CHANGE THE CODE OUTPUT. Meaning that I did not want to do anything to the code that would alter or eliminate the normal output file (or the information that would go there…). To prevent changing the output file, additional files were created to hold the purely numeric data that was desired.

In VIPRE.FOR, the following integer variables were defined.

istu1 = 18
istu2 = 14
istu3 = 15
istu4 = 16
istu5 = 17

---

[k] The script interface could possibly have been created without modifying the program source code, but it would have been much more difficult to process the program output.

[l] This is an interesting artifact of the Windows operating system. It is actually rather difficult to create an ASCII-text file without appending the '.txt' suffix. The programs MS Wordpad and MS Notepad both do it automatically. It is possible to use the MS Edit program from the command prompt to make input files, but the added functionality of the other programs is lost. The easier alternative was to make the trivial change to the source code to allow input files with the '.txt' suffix.

[m] The command is simply 'load (*filename*)" or "load *filename*". All of the numeric data is assigned to a Matlab array named *filename*.

These variables were used as file descriptors for the additional output files that were created.

```
      open(unit=istu1,file='chsum.txt',status='new',form='formatted')
      open(unit=istu2,file='chfdat.txt',status='new',form='formatted')
      open(unit=istu3,file='celldat.txt',status='new',form='formatted')
      open(unit=istu4,file='hotchnl.txt',status='new',form='formatted')
      open(unit=istu5,file='fuelthr.txt',status='new',form='formatted')
```

The corresponding code to print the output to these files is found in the subroutine 'result' in VIPRE3.FOR.

```
c STU ADDED OUTPUT ***********************************
      write(stu1,10115) i,h(i,ndxp1),tdum,rho(i,ndxp1),
     1  qlty,alfa,f(i,ndxp1),fluxd
c STU ADDED OUTPUT ***********************************
10115 format(10x,i4,2x,f10.2,4x,f10.2,5x,f9.3,1x,
     1     f9.3,7x,f9.4,1x,f11.3,1x,f10.4)

      write(stu2,10262) xdum1,xdum2,chfr(i,j),n,sflux,
     1  cchf,dflux,faxl(i,j),fgrid(i,j),fwal(i,j),g,xx

10262 format(1x,f6.1,f6.1,2x,f6.3,1x,i3,3x,1x,f7.4,3x,f7.4,
     1  5x,f7.4,2x,1x,f8.4,2x,f8.4,2x,f8.4,6x,3x,f7.4,
     2  9x,f7.4)

      write(stu4,10277) kase,pref,havg,gavg,qave,smdnbr,ihot,nhot,xhot,
     1  ghot,xx,qmeas,qpred,etime

10277 format(1x,i3,3x,f6.1,1x,f7.2,5x,f6.3,5x,f8.3,3x,
     1  f6.3,2x,i3,3x,i3,1x,f6.1,2x,f6.3,2x,f6.3,4x,f6.3,5x,
     2  f6.3,6x,f8.3)

      write(stu3,10296) xx,pres,f(ii,jj),u,g,
     1  xx1,rho(ii,jj),h(ii,jj),alfaj(ii,jj),qualij,
     3  tchff(ii,jj),qprimt

10296 format(2x,f6.1,2x,f7.2,1x,f11.3,2x,f7.2,3x,f8.3,4x,
     1  2x,f6.2,3x,f6.2,4x,f6.1,5x,f6.4,4x,f6.4,5x,f6.1,
     3  1x,f14.4)

      write(stu5,10407) htcd,trod(nclad,n,jx),tavc,trod(nclad-1,n,jx),
     1        gpcdum,trod(nfuel,n,jx),tavf,trodcl


10407   format(1x,f15.1,3x,3(f6.1,2x),f12.3,2x,3(2x,f6.1))
```

These statements were placed coincidentally with the analogous outputs to the normal output file (output.txt). It was natural to organize the changes this way. Each

output file was a rectangular array of numbers that Matlab would load into it's environment in one step.[a]

Time Step Tracking

Also placed in VIPRE.FOR was a trivial, but useful change to allow tracking of transient progression. At the end of every time step, VIPRE would print out the number of the time step that it was working on. This is comforting in the event that one is running a very long transient and one wants to know how things are going. More likely, one has an input file for a transient that continues to crash, it is a good diagnostic aid to know what time step was last processed before the program terminated.

In VIPRE.FOR, in the code just as the current time-step is complete, the following code is executed:

```
      if(stu_cnt .gt. 1) write(*,10151) stu_cnt-1
10151 format(1x,'Time step ',i5,' complete.')
      stu_cnt = stu_cnt + 1
```

Here, 'stu_cnt' is simply an integer that is initialized to zero.

Increase Maximum Number of Time Steps

In its initial condition VIPRE was capable of processing only 10 transient time steps. Since all of the transients considered in this thesis were temporally discretized into 402 time steps, a change was required.

While, for this project, this modification was performed manually, there is a tool that has been developed specifically for this purpose. The program is called SPECSET. This is a program, that takes as input a text file called 'Specs' which is listed here:

```
*ADDCOMD SPECS
   ia=3
   ib=3
   ie=60
   ip=2
   jf=5
```

[a] Incidentally, the numbers in the output file to be read by the Matlab interface must be strictly two-dimensional. By this it ismeant that if there are six space-delimited numbers in the first row, there must be six numbers in every row. This is why there are several added output files rather than only one.

```
      jg=4
      mc=530
      mf=500
      mg=915
      mi=10
      mk=5
      mn=16
      mp=50
      mr=500
      mt=10
      mx=160
      na=5
      nm=8
      lg=1
      lx=1
      lj=1
      lh=1
      mv=2
      mw=530
      mz=15
c
cend -- last common
```

Each line is interpreted by SPECSET as a variable parameter. The meaning of each variable is given in reference [48]. Once SPECSET is run, a second program, called "UPDATE" is run. This regenerates the common blocks of the VIPRE code so that the statically allocated arrays are set to be of the correct size. The source code of VIPRE is then manually updated with these new common blocks and re-compiled.

Extension of Matlab-VIPRE Interface to A Parallel Distributed Computing Environment

The development of a script-based interface and script-based coupling of multiple codes allows for relatively simple creation of fairly computationally demanding tasks. The Monte Carlo Uncertainty Procedure (MCUP), described fully in Chapter 3 of this thesis, requires many hours of computational time on even the most modern personal computers. For example, a single VIPRE transient analysis data point for the purposes of the Monte Carlo Uncertainty Procedure requires approximately 51 seconds.[o] The 2000 samples normally required for this procedure would thus consume slightly more than 28 hours of computer time.

Fortunately, the computational tasks that are required for the MCUP are highly parallelizable[p] and can thus benefit from the proliferation of computer 'clusters' built from commodity components. At the time of this writing, there are at least 14 clusters in operation at MIT including one 30-node machine within the Nuclear Engineering Department.

The purpose of this section is to provide a set of recommendations regarding the migration of the existing script-interface to a parallel computer environment. With judicious choice of software tools and parallel programming environments, the resulting package can be scalable to as large a cluster as is desired, and portable from one machine to the next. These recommendations are relevant for the VIPRE interface. The recommendations are as follows:

- Port VIPRE to the Linux environment. This assumes that the target cluster machine uses Linux as its operating system. Typically this is the case, since Linux is freely available.
- Re-write the script interface using standard, free, Linux software components. Probably the quickest route would be, to use a free spreadsheet too such as Gnumeric to hold the input data, and a scripting language such as Perl to form the core of the interface.

---

[o] A personal computer with a 1.8 GHz Pentium IV processor was used for these computations.
[p] For the MCUP, no communication is required between successive sample runs. In the parlance of the parallel programming community, the procedure is 'Embarrassingly Parallel' because it is so easy to parallelize the algorithm.

- Perl is recommended for the interface language as it has an interface to MPI, a standard parallel programming library, and an interface to Gnumeric to allow for extraction of the input data.
- The basic architecture of the interface need not change, only the components used to create each part.

This short appendix is meant to serve as a roadmap for others who may be interested in implementing such a tool. The coarse grained parallelism that exists in analysis involving the Matlab-VIPRE Interface could be exploited by relatively low-cost parallel platforms. This capability could make the more computationally demanding optimization and uncertainty analysis more realistic for routine studies.

Appendix 3 AECL-UO Look-up Table Interface

The AECL-UO Look-up table had been provided from the University of Ottawa for to allow students in MIT course 22.312 to more easily make use of this correlation.[19] The table was provided in the form of a FORTRAN 77 program where the user could place, in the source code, the plant conditions from which to compute the correlation estimated CHF. Desiring the ability to quickly and conveniently access this functionality from within a Matlab environment, a set of functions was authored to allow this access.

The basic operation of this interface is:

A wrapper functin calls a mex-file that passes the required arguments to a modified FORTRAN subroutine that performs the same task that the original program did. The FORTRAN subroutine writes the result to an ASCII text file, which is then read by the wrapper function and passed back to the Matlab workspace. The text file is then deleted.

A schematic representation of this interface is provided in Figure 43.

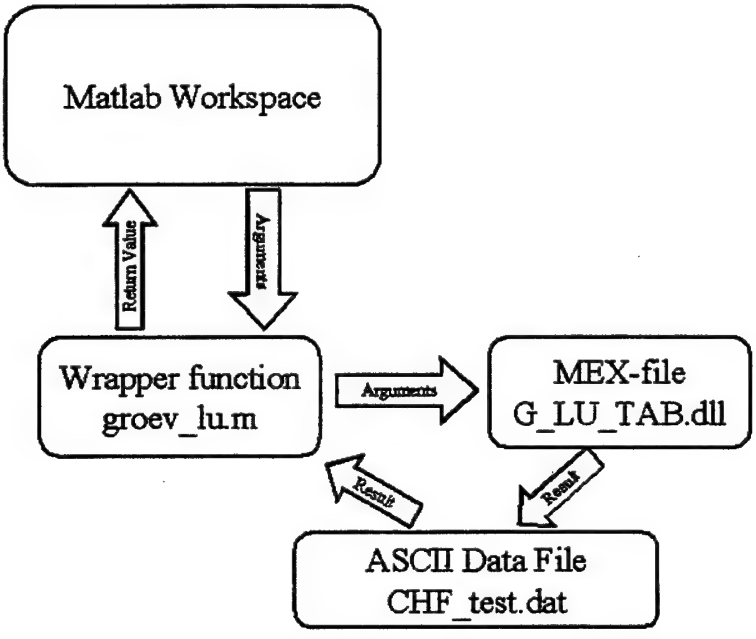


Figure 43:Schematic Representation of Matlab - AECL-UO Inteface.

The modified portions of the look-up table source code is provided here:

```
      REAL FUNCTION G_CHF(D,P,G,X)

C*****************************************************************************
C                                                                             *
C   Program Main_CHF -- Modified to a Function by                             *
C                 Stuart Blair, MIT                                           *
C   This program is used to test subroutines for                              *
C   calculation of Critical Heat Flux for                                     *
C   upward flow of water-steam inside vertical uniformly heated              *
C   round tubes using the Look-Up Table prediction method.                   *
C                                                                             *
C   Author:  Aleksandar Vasic', TRACI, Oxford Station, Ont., Canada           *
C         903255@icen.net                                                     *
C   Date:   2000 March 10                                                     *
C   Version: 0 Revision: 0                                                    *
C                                                                             *
C*****************************************************************************

C*****************************************************************************
C                                                                             *
C   Subroutines and functions called in this subroutine:                      *
C                                                                             *
C   CHF_LUT - calculate CHF values for input paramaters (D,P,G,X).            *
C                                                                             *
C*****************************************************************************

C*****************************************************************************
C                                                                             *
C   Variable declaration:                                                     *
C                                                                             *
C   Integer:                                                                  *
C   IX     : minimum node of quality in table for input value.                *
C   IG     : minimum node of mass flux in table for input value.              *
C   IP     : minimum node of pressure in table for input value.               *
C                                                                             *
C   Real:                                                                     *
C   D      : input hydraulic diameter (m).                                    *
C   P      : input pressure (kPa).                                            *
C   G      : input mass flux (kg/m2/s).                                       *
C   X      : input quality (-).                                               *
C   QCHF   : calculated critical heat flux (W/m2).                            *
C   PX     : array of table quality values (-).                               *
C   PG     : array of table mass-flux values (kg/m2/s).                       *
C   PP     : array of table pressure values (kPa).                            *
C                                                                             *
C*****************************************************************************

      REAL PX(23), PG(20), PP(21), AQCHF(23)

C*****************************************************************************
C                                                                             *
C   Call up the common blocks to pass the variables in and out of             *
C   the subroutine.                                                           *
C                                                                             *
C*****************************************************************************

      COMMON/ROWS/ PX, PG, PP, IX, IG, IP

C*****************************************************************************
C                                                                             *
C   Input parametars.                                                         *
C                                                                             *
C*****************************************************************************
      REAL D,P,G,X


C*****************************************************************************
C                                                                             *
C   Evaluate CHF values.                                                      *
C                                                                             *
```

```
C*************************************************************************

   OPEN (UNIT=1, FILE='CHF_TEST.DAT')
     CALL CHF_LUT(D,P,G,X,QCHF)
     G_CHF = QCHF
     WRITE (1,1) P,G,X,QCHF
1    FORMAT (8E12.4)
   CLOSE(1)


   END
```

The wrapper script – that is, the script that the user actually calls – is given here:

```
%groev_lu.m

function chf = gorev_lu(D,P,G,X)

addpath('C:\matlab_sv13\work\Research\CHF_CORR\Groeneveld');
G_LU_TAB(D,P,G,X);

load('CHF_TEST.DAT');

chf = CHF_TEST;

delete('CHF_TEST.DAT')
```

The remainder of the code is submitted electronically.

It is notable that this look-up function runs fairly slowly due to the requirement to communicate the result through the file system.[a] When many hundreds, or many thousands of consecutive evaluations would be required, the performance penalty is quite burdensome. In the case where the function is called relatively infrequently,[b] the poor performance is not noticeable.

[a] I.e. by writing the result to the file, then reading the file.
[b] I.e. less than one hundred times in a row.

Appendix 4 IRIS Open Matlab-Vipre Scripts

The following is a collection of high-level scripts used to perform larger, higher-level tasks. These scripts are meant as a guide to any future users of this interface to help them cobble together larger, useful scripts from the lower-level scripts presented before.

Steady State Analysis

A simple script for performing a steady-state analysis of the IRIS Open core is provided here. For this example, the W-3L correlation is used. Note that the structure of this script follows the logical pattern of: Extract input data from Excel, generate the input file, instantiate vipre, parse and store the outputs.

```
SPREADSHEET_NAME = '17x17_trans_lrss_w31.xls';
%provide access to low-level tools.
addpath('c:\matlab_sv13\work\research\vipre transient tools');
%identify and get handles to input spreadsheets
spreadsheet_filename = strcat(pwd,'\',SPREADSHEET_NAME);
[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy');
%obtain input data and generate initial input file
input_file_name = 'input_w31';
var_file_name = 'var_w31';
get_excel_input2_w31(input_sheet,var_file_name);
generate_input_file(input_file_name,var_file_name);
%close the connection to the excel spreadsheet and clean up
invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object
vipre_run_jon(input_file_name,var_file_name);
```

The last few steps of the pattern are all carried out by the script vipre_run_jon.m. Note that there are many scripts with names starting with "vipre_run_...." There are several variants because there were several versions of the Matlab-VIPRE interface – specifically, different versions of the VIPRE code to deal with. Different versions of the "vipre_run.." scripts were written to deal with those differences. The names are all different, but their functionality is the same.

Sensitivity Analysis

Correlation sensitivity was computed using two different methodologies. The first methodology focused on the degree to which the sensitivity of a given CHF correlation in

response to changes in a given parameter varied over a relatively large range in parameter value. As an example, the sensitivity of the W-3L correlation to changes in mass flux is computed for mass fluxes in the range of 1.2533 $Mlbm/ft^2 - \sec$ ± 1%.

The second methodology is used to illustrate that the sensitivity of a correlation to changes in a given parameter is a function of more than just the parameter itself. Rather, the state of other plant parameters has an influence as well. For example, the sensitivity of the W-3L correlation to changes in mass flux is computed with all five uncertain parameters in different states. What is immediately apparent from the result, is that the magnitude of the sensitivity can vary quite widely even though all of the uncertain parameters are fairly close to their nominal value. Further analysis reveals that the sensitivity is at it's worst-case value when the plant is, in some sense, closest to DNB. This occurs when mass flux and system pressure are low and core inlet temperature, linear power and $F_{\Delta H}^{E}$ are high.

These sensitivities were computed with the using the Matlab-Vipre Interface described in chapter 2 of this thesis. This interface greatly simplified the analysis by reducing the tedium involved in performing the calculation. In the case of the second methodology, the process involved several thousand VIPRE runs. The second methodology would be almost completely impractical to perform in the absence of the Matlab-Vipre Interface.

The code for the first example is given here:

```
%find_sensitivity.m
%computes the W-3L sensitivity to fluctuations in Mass Flow Rate.

PARAM_MIN = .99;
PARAM_MAX = 1.01;
NOM_PARAM = 1.2533;
FIELD = 'oper5.gin';
NUM_STEPS = 3;

spreadsheet_filename = strcat(pwd,'\','IRIS-refined VIPRE core model-17x17-np.xls');
addpath('c:\matlabr12\work\research\matlab vipre tools'); %add interface to workspace
PARAM_SPACE = linspace(PARAM_MIN,PARAM_MAX,NUM_STEPS);
PARAM_SPACE = NOM_PARAM .* PARAM_SPACE; %get the set of new values

%open the excel file
[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy');
file_root = 'flow';
i = 1;
input_file_name = strcat(file_root,num2str(i));
var_file_name = strcat(file_root,'_var',num2str(i));
```

```
get_excel_input2_w3L(input_sheet,var_file_name);
%excel input has been saved, excel activex object is no longer necessary
invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object

%perform the runs
for i = 1:NUM_STEPS
    change_input(var_file_name,FIELD,PARAM_SPACE(i));
    generate_input_file(input_file_name,var_file_name);
    vipre_run_sens(input_file_name);
    input_file_name = strcat(file_root,num2str(i+1));

end

%clean up temporary files
for i = 1:NUM_STEPS
    delete(strcat('flow',num2str(i)));
end
delete(strcat(var_file_name,'.mat'));

%collect the required data

mdnbr = zeros(NUM_STEPS,1);

for i = 1:NUM_STEPS
    direc = strcat(file_root,num2str(i),'-out');
    cd(direc);
    load channel_data;
    mdnbr(i) = corr_mdnbr;
    cd ..
end


ave_mdnbr = mean(mdnbr); %defined as the nominal MDNBR for this case -- this works ok
%since the variation in MDNBR has been shown by computational experiment to be
%very nearly linear over a 1% perturbation in parameter range (at least for temperature)

percent_var_mdnbr = ((max(mdnbr) - min(mdnbr))/ave_mdnbr) * 100;

percent_var_parameter = (PARAM_MAX - PARAM_MIN) * 100;

sensitivity = percent_var_mdnbr/percent_var_parameter;
```

Code for the second methodology is provided here. The driver script is:

```
%sensitivity.m

mass_flux = [1.1906,1.25326,1.3159];
lin_power = [2.9792,3.04,3.1];
inlet_temp = [553.6,557.6,561.6];
system_pressure = [2230,2259.39,2290];
f_dh_e = [1.0,1.02,1.04];

%generate permutation matrices
index = 1;
for m = 1:3
    for l = 1:3
        for i = 1:3
            for s = 1:3
                for f = 1:3
                    perm_list(index,:) = [m,l,i,s,f];
                    index = index + 1;
                end
            end
        end
    end
end
```

```
[num_permutes,n] = size(perm_list);
%this permutation list will have to be cycled through 5 times
%at each cycle, I will take the sensitivity with respect to a different parameter

sensitivity_array = zeros(num_permutes,5);

spreadsheet_filename = strcat(pwd,'\','IRIS-refined VIPRE core model-17x17-np.xls');
addpath('c:\matlabr12\work\research\matlab vipre tools');


%open the excel file
[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy');
file_root = 'flow';
i = 1;
input_file_name = strcat(file_root,num2str(i));
var_file_name = strcat(file_root,'_var',num2str(i));
get_excel_input2_w3L(input_sheet,var_file_name);
%excel input has been saved, excel activex object is no longer necessary
invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object

%now all of the pieces are in place -- start with mass flux
old_value = 1.0; %for change_f_dh_e.m  -- using nominal inputs -> the existing value is
1.0
index = 0;

mass_flux_sensitivity;
linear_power_sensitivity;
inlet_temperature_sensitivity;
system_pressure_sensitivity;
f_dh_e_sensitivity;
```

Notice, here at the end, five scripts are called to get the sensitivity of the correlation with respect to mass flux, linear power, etc… These scripts are nearly identical, the mass_flux_sensitivity script is provided for illustration:

```
%mass_flux_sensitivity.m

for i = 1:num_permutes
    index = index + 1;
    sprintf('Commencing Run # %d',index)
    PARAM_MIN = .99;
    PARAM_MAX = 1.01;
    NOM_PARAM = mass_flux(perm_list(i,1));
    FIELD = 'oper5.gin';
    NUM_STEPS = 3;
    PARAM_SPACE = linspace(PARAM_MIN,PARAM_MAX,NUM_STEPS);
    PARAM_SPACE = NOM_PARAM .* PARAM_SPACE; %get the set of new values
    %change the other parameters
    change_input(var_file_name,'oper5.pwrinp',lin_power(perm_list(i,2)));
    change_input(var_file_name,'oper5.hin',inlet_temp(perm_list(i,3)));
    change_input(var_file_name,'oper5.pref',system_pressure(perm_list(i,4)));
    change_f_dh_e(var_file_name,old_value,f_dh_e(perm_list(i,5)));
    old_value = f_dh_e(perm_list(i,5)); %update this number for future use

    temp_mdnbr = zeros(NUM_STEPS,1); %storage of the mdnbr results for each run in
    %the sensitivity analysis

    for j = 1:NUM_STEPS
    change_input(var_file_name,FIELD,PARAM_SPACE(j));
    generate_input_file(input_file_name,var_file_name);
    vipre_run_stu_jac1(input_file_name,var_file_name);
```

```
    load mdnbr_dat;
    temp_mdnbr(j) = corr_mdnbr;
    end

    ave_mdnbr = mean(temp_mdnbr); %defined as the nominal MDNBR for this case -- this
works ok
    %since the variation in MDNBR has been shown by computational experiment to be
    %very nearly linear over a 1% perturbation in parameter range (at least for
temperature)

    percent_var_mdnbr = ((max(temp_mdnbr) - min(temp_mdnbr))/ave_mdnbr) * 100;

    percent_var_parameter = (PARAM_MAX - PARAM_MIN) * 100;

    sensitivity = percent_var_mdnbr/percent_var_parameter;
    sensitivity_array(i,1) = sensitivity; %store this in the array
end
```

Transient Analysis

An example script for a single transient simulation run is provided here.

```
%single_trans_run_w31.m

SPREADSHEET_NAME = '17x17_trans_lrss_w31.xls';
%provide access to low-level tools.
addpath('c:\matlab_sv13\work\research\vipre transient tools');
%identify and get handles to input spreadsheets
spreadsheet_filename = strcat(pwd,'\',SPREADSHEET_NAME);
[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy_trans');
transient_sheet = get_worksheet(spread_sheet,'trans_input');
%obtain input data and generate initial input file
input_file_name = 'input_w31';
var_file_name = 'var_w31';
get_excel_input2_trans_w31(input_sheet,transient_sheet,var_file_name);
generate_input_file_trans(input_file_name,var_file_name);
%close the connection to the excel spreadsheet and clean up
invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object
vipre_run_jon(input_file_name,var_file_name);
```

Monte Carlo Uncertainty Analysis

The basic procedure for performing the Monte Carlo Uncertainty Procedure (MCUP) is outlined in Chapter 3. This appendix will provide a sample of the Matlab-VIPRE Interface scripts needed to accomplish this task.

The presented script is for performing the MCUP for the PLOFA transient using the EPRI CHF correlation. For this analysis, 4000 transient sample runs will be taken. Each uncertain parameter will be represented by a rectangular probability distribution. The results will be collected and then analyzed with a series of scripts.

The source code is given as follows:

```
%trans_mc_plofa_epri.m

SPREADSHEET_NAME = '17x17_trans_plofa_epri.xls';
NUM_SAMPLES = 4000;
%=========================================================================================
%================== Parameter Information ================================================
%=========================================================================================

%this section includes the parameters to be sampled, along with their statistical
%distribution.  I think to start, I will just employ a rectangular distribution
%for all parameters.  But this should be easily adjusted.

inlet_temp = zeros(NUM_SAMPLES,1);
inlet_temp_field = 'oper5.hin';
system_pressure = zeros(NUM_SAMPLES,1);
system_pressure_field ='oper5.pref';
mass_flux = zeros(NUM_SAMPLES,1);
mass_flux_field = 'oper5.gin';
linear_power = zeros(NUM_SAMPLES,1);
linear_power_field = 'oper5.pwrinp';
f_dh_e = zeros(NUM_SAMPLES,1);

%=========================================================================================
%================ Data Array =============================================================
%=========================================================================================
mdnbr = zeros(NUM_SAMPLES,1);



%=========================================================================================
%============== Parameter Random Sample Generation =======================================
%=========================================================================================



%================================================
%========= Inlet Temperature ====================
%================================================

%rectangular distribution at 557.6 +/- 10 degrees F
NOMINAL_VALUE = 557.6;
LOWER_RANGE = 4;
UPPER_RANGE = 4;

%generate rectanular distribution about the nominal value
inlet_temp =  (UPPER_RANGE + LOWER_RANGE) .* rand(NUM_SAMPLES,1) + ...
    (NOMINAL_VALUE - LOWER_RANGE);

%================================================
%================================================

%================================================
%========= System Pressure ======================
%================================================

NOMINAL_VALUE = 2259.39;
LOWER_RANGE = 30;
UPPER_RANGE = 30;

%generate rectangular distribution about the nominal value
system_pressure =  (UPPER_RANGE + LOWER_RANGE) .* rand(NUM_SAMPLES,1) + ...
    (NOMINAL_VALUE - LOWER_RANGE);

%================================================
%================================================

%================================================
%========= Mass Flux ============================
```

```
%================================================

NOMINAL_VALUE = 1.25326;
LOWER_RANGE = NOMINAL_VALUE*(.02);
UPPER_RANGE = NOMINAL_VALUE *(.02);

%generate rectangular distribution about the nominal value
mass_flux =  (UPPER_RANGE + LOWER_RANGE) .* rand(NUM_SAMPLES,1) + ...
    (NOMINAL_VALUE - LOWER_RANGE);

%================================================
%================================================

%================================================
%========== Linear Power ========================
%================================================

NOMINAL_VALUE = 3.040;
LOWER_RANGE = NOMINAL_VALUE*(.02);
UPPER_RANGE = NOMINAL_VALUE*(.02);

%generate rectangular distribution about the nominal value
linear_power =  (UPPER_RANGE + LOWER_RANGE) .* rand(NUM_SAMPLES,1) + ...
    (NOMINAL_VALUE - LOWER_RANGE);

%================================================
%================================================

%================================================
%========== F_dH_E ==============================
%================================================

NOMINAL_VALUE = 1.02;
LOWER_RANGE = .02;
UPPER_RANGE = .02;

%generate rectangular distribution about the nominal value
f_dh_e =  (UPPER_RANGE + LOWER_RANGE) .* rand(NUM_SAMPLES,1) + ...
    (NOMINAL_VALUE - LOWER_RANGE);

%================================================
%================================================

%=============================================================================
%============== Get Tools and Initialize Input ===============================
%=============================================================================

%provide access to low-level tools.
addpath('c:\matlab_sv13\work\research\vipre transient tools');

%identify and get handles to input spreadsheets
spreadsheet_filename = strcat(pwd,'\',SPREADSHEET_NAME);
[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
%set(xl,'Visible','False');
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy_trans');
transient_sheet = get_worksheet(spread_sheet,'trans_input');

%obtain input data and generate initial input file
file_root = 'input';
input_file_name = file_root;
var_file_name = strcat(file_root,'var');
get_excel_input2_trans_epri(input_sheet,transient_sheet,var_file_name);
generate_input_file_trans(input_file_name,var_file_name);

%close the connection to the excel spreadsheet and clean up
invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object

%=============================================================================
%============== Run the Simulations ==========================================
```

```
%=========================================================================

for i = 1:NUM_SAMPLES
   sprintf('Commencing Simulation Run # %d',i)

   if i == 1
      fdhe_old = 1;
   else
      fdhe_old = f_dh_e(i-1);
   end

    change_input(var_file_name,inlet_temp_field,inlet_temp(i));
    change_input(var_file_name,system_pressure_field,system_pressure(i));
    change_input(var_file_name,mass_flux_field,mass_flux(i));
    change_input(var_file_name,linear_power_field,linear_power(i));
    change_fdhe(var_file_name,fdhe_old,f_dh_e(i));
    generate_input_file_trans(input_file_name,var_file_name);
    vipre_run_sv5a_mc(input_file_name);
    load mdnbr_dat;
    mdnbr(i) = min(corr_mdnbr);
end

save run_dnb_data mdnbr;
save sample_data inlet_temp system_pressure mass_flux linear_power f_dh_e;
```

At this point, the transient runs have all been made. The MDNBR results from each sample is saved to a file, as are the random samples of the parameters.

The scripts to perform the data analysis are provided here:

```
%data analysis

TOTAL_RUNS = 4000; %total number of runs to be analyzed

tot_mdnbr(1:TOTAL_RUNS) = 0; %initialize the data array

cd('4000 Runs 15 June 2003');
load run_dnb_data
tot_mdnbr(1:4000) = mdnbr;
cd ..
%analyze first 4000 runs
[low_conf_mean,high_conf_std] = mc_dist_analysis(tot_mdnbr(1:4000));
y4000 = normal_dist(low_conf_mean,high_conf_std,X);
mean4000 = low_conf_mean - 1.65*high_conf_std
%look at the overall effect
SKIP = 5;

[n,bin_cent] = hist(tot_mdnbr,25);
n = n .* (1/(length(tot_mdnbr)*(bin_cent(2) - bin_cent(1))));
bar(bin_cent,n);
hold on;
plot(X(1:SKIP:end),y4000(1:SKIP:end),'hc');
xlabel('MDNBR');
ylabel('Probability Density');
title('PLOFA - Bowring MDNBR Distributions');
```

Two important utility functions are used here, the first, mc_dist_analysis.m , accepts the MDNBR sample values as data, and returns the 95 percent lower confidence mean and 95 percent upper confidence standard deviation.

```
%mc_dist_analysis.m

function [lc_mean,hc_std] = mc_dist_analysis(data)
[m,n] = size(data');
dof = m-1;
%get a handle to the excel workbook functions
ex = actxserver('excel.application');
%set(ex,'visible','false');
func = get(ex,'worksheetfunction');

samp_mean = mean(data(1:m));
samp_std = std(data(1:m));

if (dof < 1000)
   chi = invoke(func,'chiinv','0.95',num2str(dof));
   upp_conf_std = samp_std*sqrt(dof/chi);
else
   upp_conf_std = samp_std;
end

low_conf_mean = samp_mean -
invoke(func,'confidence','0.1',num2str(upp_conf_std),...
   num2str(m));

lc_mean = low_conf_mean;
hc_std = upp_conf_std;

delete(ex);
```

Microsoft Excel workbook functions 'chiinv' and 'confidence' are used in this procedure to eliminate the requirement to program these rather complicated functions by hand.

The Matlab diary results for this run are:

low_conf_mean =

1.4353

high_conf_std =

0.0282

mean4000 =

1.3888

Appendix 5 Steady State CPR Calculations

This appendix presents the scripts that were used to generate the steady-state CPR vs. MDNBR and MDNBR vs. Power curves shown in Chapter 3.

MDNBR vs Power

This is a very straight-forward algorithm that is implemented with only a few lines of Matlab code for each correlation included in the plot. Most of the Matlab code is dedicated to producing an attractive graphical output.

The high-level driver script is given here:

```
%power_vary_all.m
% obtain the mdnbr vs. power data for each correlation
 power_range_macb;
 pow_it = zeros(20,4);
 pow_it(:,1) = power';
 mdnbr_it(:,1) = mdnbr_p;
%
 power_range_baw2;
 pow_it(:,2) = power';
 mdnbr_it(:,2) = mdnbr_p;
%
 power_range_w3l;
 pow_it(:,3) = power';
 mdnbr_it(:,3) = mdnbr_p;
%
 power_range_groen;
 pow_it(:,6) = power';
 mdnbr_it(:,6) = mdnbr_p;
%
 power_range_epri;
 pow_it(:,4) = power';
 mdnbr_it(:,4) = mdnbr_p;
%
 power_range_bowr
 pow_it(:,5) = power';
 mdnbr_it(:,5) = mdnbr_p;
% plot the data
x = linspace(min(pow_it(1,:)),max(pow_it(end,:)),1000);
plot(pow_it(:,1),mdnbr_it(:,1),'-og','LineWidth',2);
hold on
plot(pow_it(:,2),mdnbr_it(:,2),'-+k','LineWidth',2);
plot(pow_it(:,3),mdnbr_it(:,3),'-sc','LineWidth',2);
plot(pow_it(:,6),mdnbr_it(:,6),'-*r','LineWidth',2);
plot(pow_it(:,4),mdnbr_it(:,4),'-dm','LineWidth',2);
plot(pow_it(:,5),mdnbr_it(:,5),'-xb','LineWidth',2);
plot(x,1.3,'-r','LineWidth',2)
title('MDNBR vs Power')
xlabel('Linear Power (kw/ft)')
ylabel('MDNBR')
legend('MacBeth','B&W-2','W-3L','AECL-UO Look-up Table','EPRI','Bowring','MDNBR= 1.3')

grid on
```

The scripts used to obtain the MDNBR vs. power data are straight-forward as well. Two examples are given here[c]:

power_range_groen.m

```
%==============get input data=======================================
spreadsheet_filename = strcat(pwd,'\','17x17_trans_lrss_w3l.xls');
addpath('c:\matlab_sv13\work\research\vipre transient tools');

[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy');

var_filename = 'w3l_input_dat';
input_filename='w3l_input';
get_excel_input2_w3l(input_sheet,var_filename); %extract input data from excel.
%script not provided due to length...

invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object
%=========================================================================
FIELD= 'oper5.pwrinp';
% MIN_POWER = 3.0; %kw/ft
% MAX_POWER = 4.75; %kw/ft
% NUM_STEPS = 20;

power = linspace(MIN_POWER,MAX_POWER,NUM_STEPS);
mdnbr_p = zeros(NUM_STEPS,1);


for i = 1:NUM_STEPS
    change_input(var_filename,FIELD,power(i));
    generate_input_file(input_filename,var_filename);
    vipre_run_jon(input_filename,var_filename);
    load('channel_data');
    %here's where I need to snag the data
    load(var_filename) % get the input data in my context
    [hot_chnl_type,j] = find(geom6 == corr_hot_channel);
    flow_area = geom5(hot_chnl_type).carea;
    heated_perim = geom5(hot_chnl_type).cph;

    qual = channel_equilibrium_quality(1,:,corr_hot_channel);
    mass_flux = channel_mass_flux(corr_hot_channel); %mlbm/hr - ft^2
    system_press = oper5.pref; %psia
    D_hyd = 4*flow_area/heated_perim; %this is in inches - must be converted to meters.
    local_press =  channel_delta_p_local(1,:,corr_hot_channel) + system_press; %psia
    loc_heat_flux = channel_surface_hf(1,:,corr_hot_channel); %btu/ft^2-hr

    %now - convert
    D_hyd = D_hyd * (1/39.37); %m
    local_press = local_press .* (6.8947); %kPa
    mass_flux = mass_flux * (1356.222); %kg/m^2-sec
    loc_heat_flux = loc_heat_flux .* (0.31546e7); % W/m^2
    axial_zones = min(length(qual),length(local_press));
    chf_groen_local = zeros(axial_zones,1);
    dnbr_groen_local = zeros(axial_zones,1);
    for k = 1:axial_zones
        chf_groen_local(k) = groev_lu(D_hyd,local_press(k),mass_flux,qual(k));
        %modify by the appropriate factor
        %k1
        k1 = 1;
        if (D_hyd > 0.002) & (D_hyd < 0.016)
```

[c] It would be useful to see the script used to obtain MDNBR vs Power data for the AECL-UO correlation as well as a correlation that is built-in to VIPRE – if only to highlight the added logic required for the AECL-UO correlation..

```
            k1 = (0.008/D_hyd)^(1/3);
        elseif (D_hyd > 0.016)
            k1 = 0.79;
        else
            k1 = 1.0;
        end

        %other factors assumed equal to 1 for now...
        chf_groen_local(k) = chf_groen_local(k)*k1;



        if(loc_heat_flux(k) > 0)
            dnbr_groen_local(k) = chf_groen_local(k)/loc_heat_flux(k);
        else
            dnbr_groen_local(k) = 10.0;
        end
    end


    mdnbr_p(i) = min(dnbr_groen_local);
end
```

power_range_epri.m

```
%==============get input data=====================================
spreadsheet_filename = strcat(pwd,'\','17x17_trans_clofa_epri.xls');
addpath('c:\matlab_sv13\work\research\vipre transient tools');

[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy');

var_filename = 'epri_input_dat';
input_filename='epri_input';
get_excel_input2_epri(input_sheet,var_filename); %extract input data from excel.
%script not provided due to length...

invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object
%=========================================================================
FIELD= 'oper5.pwrinp';
TEMP_FIELD = 'oper5.hin';
MIN_POWER = 3.0; %kw/ft
MAX_POWER = 5.5; %kw/ft
NUM_STEPS = 20;
NOM_POWER = 3.04; %kw/ft
NOM_INLET_TEMP = 557.6; % degrees F
NOM_DELTA_T = 67.0; %degrees F

power = linspace(MIN_POWER,MAX_POWER,NUM_STEPS);
new_Tin = NOM_INLET_TEMP .* (ones(NUM_STEPS,1)) - (0.5).* ((power' -
NOM_POWER)./NOM_POWER).*NOM_DELTA_T;
mdnbr_p = zeros(NUM_STEPS,1);

for i = 1:NUM_STEPS

    change_input(var_filename,FIELD,power(i));
    change_input(var_filename,TEMP_FIELD,new_Tin(i));
    generate_input_file(input_filename,var_filename);
    vipre_run_jon(input_filename,var_filename);
    load('channel_data');
    mdnbr_p(i) = corr_mdnbr;
end
```

CPR vs MDNBR

The creation of this curve was somewhat more complicated. The goal was to produce a plot where each correlation under consideration would be varied over a uniform range of power ratios. To accomplish this, one must know what linear power (for the given power shape) produces the condition of MDNBR = 1. This was computed with the first script presented. The second script cpr_vary_all.m uses :

power_ratio_epri.m

```
%power_ratio_epri.m

%==============get input data=======================================
spreadsheet_filename = strcat(pwd,'\','17x17_trans_clofa_epri.xls');
addpath('c:\matlab_sv13\work\research\vipre transient tools');

[xl,spread_sheet] = get_spreadsheet(spreadsheet_filename);
input_sheet = get_worksheet(spread_sheet,'input-nom-dummy');

var_filename = 'epri_input_dat';
input_filename='epri_input';
get_excel_input2_epri(input_sheet,var_filename); %extract input data from excel.
%script not provided due to length...

invoke(spread_sheet,'Save');
invoke(spread_sheet,'Close'); %close the file that I was working with
delete(xl);%delete the activex server object
%====================================================================
```

Here a simple Bisection algorithm is used to find the power that produces MDNBR = 1 (with a tolerance on MDNBR specified at 0.0001).

```
FIELD= 'oper5.pwrinp';
NOMINAL_POWER = 3.04;
MIN_RANGE = 4.0;
MAX_RANGE = 5.2;
MAX_ITER = 100;
TARGET = 1.0;
TOL = 1e-4;
iter = 0;
cont = 1;
pow = MIN_RANGE;

while(cont==1)
    iter = iter + 1
    change_input(var_filename,FIELD,pow);
    generate_input_file(input_filename,var_filename);
    vipre_run_jon(input_filename,var_filename);
    load('channel_data');
    mdnbr_p = corr_mdnbr;

    if (mdnbr_p < (TARGET - TOL)) %reduce power
        MAX_RANGE = pow;
        pow = 0.5*(pow + MIN_RANGE);
    elseif (mdnbr_p > (TARGET+TOL)) %increase power
        MIN_RANGE = pow;
        pow = 0.5*(pow+MAX_RANGE);
    else
        %desired power
```

```
        sprintf('Predicted Power for CHF = %d',pow)
        sprintf('Predicted CPR = %d',pow/NOMINAL_POWER)
        cont = 0;
    end

    if(iter > MAX_ITER)
        'iteration failed!!!'
        cont = 0;
    end

end
```

A version of this script is written for each correlation. The output is a very close approximation of the power where MDNBR=1 for each correlation.

Armed with this data, a series of VIPRE runs had to be made to find the MDNBR that corresponds to a set of specified CPR points (for the plot provided, CPR is to vary – for each correlation – between 2.0 and 0.98). This gives the MDNBR for each specified CPR for each correlation, and hence the desired plot can be made.

cpr_vary_all.m

```
%ordered list of linear heat-rates that result in MDNBR = 1.
%[MacBeth,B&W-2,W-3L,EPRI,Bowring,AECL-UO]
pow_one = [5.853,4.32,4.571,4.7547,4.871,4.342];


MAX_CPR = 2.0;
MIN_CPR = 0.98;
NUM_STEPS = 20;

pow_max = pow_one .* (1/MIN_CPR);
pow_min = pow_one .* (1/MAX_CPR);

pow_it = zeros(NUM_STEPS,6);
mdnbr_it = zeros(NUM_STEPS,6);
```

These are slightly modified versions of the same functions presented previously – the scripts used in power_vary_all.m are just re-formatted slightly to behave as functions. Logically there is no difference.

```
[pow_it(:,1),mdnbr_it(:,1)]= power_range_macb(pow_min(1),pow_max(1),NUM_STEPS);
 [pow_it(:,2),mdnbr_it(:,2)]=power_range_baw2(pow_min(2),pow_max(2),NUM_STEPS);
 [pow_it(:,3),mdnbr_it(:,3)]=power_range_w31(pow_min(3),pow_max(3),NUM_STEPS);
 [pow_it(:,6),mdnbr_it(:,6)]=power_range_groen(pow_min(6),pow_max(6),NUM_STEPS);
 [pow_it(:,4),mdnbr_it(:,4)]=power_range_epri(pow_min(4),pow_max(4),NUM_STEPS);
 [pow_it(:,5),mdnbr_it(:,5)]=power_range_bowr(pow_min(5),pow_max(5),NUM_STEPS);

for i=1:6
    cpr_it(:,i) = (pow_it(:,i) .* (1/pow_one(i))) .^ (-1);
end
```

Here, I would save the data, so that if the plot needed to be re-formatted in any way, the data would still be available.[d]

```
save cpr_data_file pow_it mdnbr_it cpr_it

figure

plot(cpr_it(:,2),mdnbr_it(:,2),'-+k','LineWidth',2);
hold on
plot(cpr_it(:,1),mdnbr_it(:,1),'-og','LineWidth',2);
plot(cpr_it(:,3),mdnbr_it(:,3),'-sc','LineWidth',2);
plot(cpr_it(:,6),mdnbr_it(:,6),'-*r','LineWidth',2);
plot(cpr_it(:,5),mdnbr_it(:,5),'-xb','LineWidth',2);
plot(cpr_it(:,4),mdnbr_it(:,4),'-dm','LineWidth',2);
set(get(gcf,'CurrentAxes'),'XDir','reverse') %makes the XDir increasing right to left
title('MDNBR vs Critical Power Ratio','FontSize',12,'FontWeight','bold')
xlabel('CPR','FontSize',12)
ylabel('MDNBR')
legend('B&W-2','MacBeth','W-3L','AECL-UO Look-up Table','Bowring','EPRI')
grid on
```

[d] The computational time required to generate the data required for the plot was not large, but not insignificant either. 120 VIPRE runs were used to create all of the interior points. This takes a couple of minutes, and it is worth-while to save the data while you are fine-tuning the plot.

Appendix 6 Uncertainty Analysis Sample Calculations

Improved Thermal Design Procedure

The Improved Thermal Design Procedure (ITDP) was described fully in Chapter 3. In this appendix, a complete sample calculation will be made to determine the $DNBR_{ITDP}$ for the W-3L correlation. The uncertain parameters, and the the associated W-3L correlation sensitivities are provided in Table 18 for reference. For illustration, all uncertain parameters will be assumed to have a rectangular distribution.

Parameter	Mean Value	Range	W-3L Sensitivity
Core Inlet Temperature	557.6 °F	± 4 °F	3.65
Coolant Mass Flux	$1.235\frac{MLbm}{Hr}$	± 5 %	0.778
Core Linear Power	$3.04\frac{KW}{ft}$	± 2 %	1.601
System Pressure	2259.4 psia	± 30 psia	0.777
$F_{\Delta H}^{E}$	1.02	± .02	0.595

Table 18: Parameter Distributions and W-3L Sensitivities.

The primary intermediate result will be to compute σ_y^2 . As shown in Chapter 3, this value can be expressed as follows:

$$\sigma_y^2 = \mu_y^2 s_1^2 \left(\frac{\sigma_1}{\mu_1}\right)^2 + \mu_y^2 s_2^2 \left(\frac{\sigma_2}{\mu_2}\right)^2 + \cdots + \mu_y^2 s_n^2 \left(\frac{\sigma_n}{\mu_n}\right)^2$$

For this application there are five parameters, therefore n = 5. The mean values are listed above, using the assumption that the parameters are rectangularly distributed across the ranges listed, $\{\sigma_1,\ldots,\sigma_5\}$ can computed from

$$\sigma^2 = \frac{(b-a)^2}{12}$$

Core Inlet Temperature: $\sigma_1^2 = \frac{(561.6-553.6)^2}{12} = 5.33$

Mass Flux: $\sigma_2^2 = \frac{[1.253(1.05-0.95)]^2}{12} = 0.001308$

Core Linear Power: $\sigma_3^2 = \frac{[3.04(1.02-0.98)]^2}{12} = 0.001232$

System Pressure: $\sigma_4^2 = \frac{(2289.4-2229.4)^2}{12} = 300$

$F_{\Delta H}^E$: $\sigma_5^2 = \frac{(1.04-1.0)^2}{12} = 0.0001333$

Using the above computed values with the given values for $\{\mu_1,\ldots,\mu_5\}$, $\{s_1,\ldots,s_5\}$ and that μ_y is by construction unity, the expression for σ_y^2 becomes:

$$\sigma_y^2 = (3.65)^2 \frac{5.33}{(557.6)^2} + (0.778)^2 \frac{0.001308}{(1.253)^2} + \cdots$$
$$(1.601)^2 \frac{0.001232}{(3.04)^2} + (0.777)^2 \frac{300}{(2259.4)^2} + (0.595)^2 \frac{0.0001333}{(1.02)^2} =$$
$$10^{-4} \cdot (2.284 + 5.044 + 3.417 + 0.354 + 0.454) = 11.554 \times 10^{-4}$$

The last line is presented explicitly to allow one, at a glance, to determine which parameters weigh most heavily on the total uncertainty of DNBR.

Now we must find the factor F_u:

$$F_u = \mu_y - 1.645 \cdot \sigma_y = 1 - 1.645 \cdot \sigma_y = 1 - 1.645\sqrt{11.554 \times 10^{-4}} = 1 - 0.05591 = 0.9441$$

And $DNBR_{ITDP}$ can now be found as:

$$DNBR_{ITDP} = \frac{DNBR_{correlation-limit}}{F_u} = \frac{1.3}{0.9441} = 1.377$$

Monte Carlo Uncertainty Procedure

The Monte Carlo Uncertainty Procedure (MCUP) was discussed in detail in Chapter 3. The appendix provides a detailed sample calculation for the LR/SS casualty using the W-3L correlation. For this analysis, 2000 transient runs were completed and, for each run, the MDNBR was recorded. The analysis was completed using a set of Matlab scripts as shown in a previous appendix. For the sake of illustration, I will perform the analysis using only the first 10 transient runs so that the calculations can be shown explicitly.

The resulting MDNBR for the first 10 runs are:

2.1600
2.1430
2.0910
2.1570
2.1960
2.1050
2.1900
2.1530
2.2130
2.1420

The sample mean is given by:

$$\hat{\mu}_{MDNBR} = \frac{1}{N}\sum_{i=1}^{N} MDNBR_i = 2.155$$

The sample standard deviation is given by:

$$\hat{\sigma}_{MDNBR} = \frac{1}{(N-1)}\left[\sum_{i=1}^{N} MDNBR_i^2 - N\mu_{MDNBR}^2\right]^{1/2} = 0.0383$$

Next, the sample standard deviation is shifted up to the 95% one-sided upper probability value. This is done using the χ^2 Distribution with n-1=10-1=9 degrees of freedom. The result is $\chi^2 = 3.3251$.[a] The sample standard deviation is adjusted as follows:

$$\sigma_{MDNBR} = \hat{\sigma}_{MDNBR}\left[\frac{n-1}{\chi^2}\right]^{1/2} = 0.0383\left[\frac{9}{3.3251}\right]^{0.5} = 0.0630$$

Note that since there were so few samples taken, the required confidence limit imposes a dramatic adjustment on the sample standard deviation. For larger samples of size, for example, 1000, the sample standard deviation is virtually unchanged.[b]

Now, using the 95% confidence standard deviation, the sample mean is adjusted to the 95% one-sided lower probability value. This is done using the Student's *t* Distribution as follows:

$$\mu_{MDNBR} = \hat{\mu}_{MDNBR} - \frac{t\sigma_{MDNBR}}{\sqrt{n}}$$

The *t* value can be obtained from a standard table. The result is:

$$\mu_{MDNBR} = 2.1550 - (1.645)\frac{0.0630}{\sqrt{10}} = 2.1222$$

The μ_{MDNBR} is then compared to the correlation limit MDNBR for the W-3L correlation (1.3) to verify that thermal limits are not exceeded.

---

[a] Actually, the value given is the result of the MS Excel workbook function CHIINV which numerically inverts the χ^2 distribution. The result read off from a table given in reference [24] gives the value $\chi^2 = 3.33$.

[b] As discussed earlier, for n > 1000, this adjustment is not even made.

Appendix 7 FRAPCON Interface

Code for the Matlab-FRAPCON Interface is presented in this appendix. All of the data for the input variables exposed by this interface is entered directly into a Matlab-style script. An example is provided here:

FRAPCON Input Data Collection

The input data itself is provide here:

```
%frap_data.m

%FRPCN section start
im=71; %number of time steps
na=12; % numer of equal-length axial regions along the rods
mechan=2; %not sure what this means
ngasr=15; %number of equal volume rings in pellet for gas release calculations
%FRPCN section end

%FRPCON section start
cpl=24.00; %cold plenum length in inches
crdt=0.0; %initial thickness of crud layer on cladding outside surface in mils
thkcld=0.034; %cladding wall thickness in inches
thkgap=0.0045; %pellet-cladding as-fabricated gap thickness in inches
dco=0.5512; %cladding outer diameter -- in inches
pitch=0.5712; %center-to-center distance between rods in a square array (inches)
den=96.0;%as-fabricated apparent fuel density
dspg=0.3;%outer diameter of plenum spring (inches)
fa=1.55; %peak-to-average power ratio for cosine type axial power distribtution
dspgw=0.04; %Diameter of plenum spring wire (inches)
enrch=9.98; %fuel pellet u-235 enrichment
fgpav=14.7; %initial fill-gas pressure
hdish=0.0; % height of pellet dish, assumed to be a spherical indentation (inches)
hplt=0.5; % height of each pellet (inches)
icm=4; %cladding type indicator 2= zirc 2, 4 = zirc 4
gadoln=2.5; %weight fraction of gadolinia in urania-gadolinia fuel pellets
icor=2; %index for curd model 0,1=constant crud , 2=variable crud-- growth rate is CRDTR
idxgas=1; %initial fill-gas type indicator 1=He, 2=air, 3=nitrogen, 4=fission gas,
5=argon
iplant=-2; %signal for which type of reactor: -2=pwr, -3=bwr, -4=hbwr
iq=1;  %Indicator for axial power shape -- 1 = chopped cosine, 0 = user defined
%ivoid=1;
jdlpr=0; %output print control: 0=all axial odes, 1=peak-power axial node
totl=14.0; %the total (active) fuel column length
roughc=1.97e-5; %the cladding surface arithmetic mean roughnes, peak to average (inches)
roughf=8.3e-5; %the fuel pellet surface arithmetic mean roughness, peak to average
(inches)
vs=100.0; %number of turns in the plenum spring
nunits=1; %signal for unit system to be used for input and output 1=british, 0=si
(warning for british units)
rsntr=150.0; %the increase in pellet density expected during in-reactor operation (kg/m3)
nsp=0; %signal for time-dependent input arrays (P2,TW,GO) 0=single values, 1=each time
step
p2=2250.0; %pressure (psia)
tw=557.6; %coolant inlet temperature. (F)
go=1.27e6; %mass-flux of coolant around fuel rod.  (lb/hr-ft2)
qmpy=2.0; %linear heat rate during each time step -- in kw/ft
time=41;  %number of days per time step
slim = 0.05; %limit on swelling -- volume percent
```

```
%FRPCON section end
```

This data is called into the Matlab workspace context, and the data structures required for input file generation are provided with the following script. The only required input is a character string representing the desired name of the input variable file:

```
function frap_input_data(var_file_name)

frap_data;
frap_title='This is the title';
out_que = char('frap_title');
out_que = char(out_que,'im');
out_que = char(out_que,'na');
out_que = char(out_que,'mechan');
out_que = char(out_que,'ngasr');
out_que = char(out_que,'cpl');
out_que = char(out_que,'crdt');
out_que = char(out_que,'thkcld');
out_que = char(out_que,'thkgap');
out_que = char(out_que,'dco');
out_que = char(out_que,'pitch');
out_que = char(out_que,'den');
out_que = char(out_que,'dspg');
out_que = char(out_que,'fa');
out_que = char(out_que,'dspgw');
out_que = char(out_que,'enrch');
out_que = char(out_que,'fgpav');
out_que = char(out_que,'hdish');
out_que = char(out_que,'hplt');
out_que = char(out_que,'icm');
out_que = char(out_que,'gadoln');
out_que = char(out_que,'icor');
out_que = char(out_que,'idxgas');
out_que = char(out_que,'iplant');
out_que = char(out_que,'iq');
out_que = char(out_que,'jdlpr');
out_que = char(out_que,'totl');
out_que = char(out_que,'roughc');
out_que = char(out_que,'roughf');
out_que = char(out_que,'vs');
out_que = char(out_que,'nunits');
out_que = char(out_que,'rsntr');
out_que = char(out_que,'nsp');
out_que = char(out_que,'p2');
out_que = char(out_que,'tw');
out_que = char(out_que,'go');
out_que = char(out_que,'qmpy');
out_que = char(out_que,'time');
out_que = char(out_que,'slim');


out_que = cellstr(out_que);


%save the input variables to file to allow further manipulation
%var_file_name = 'input_vars';
%check to see if the file currently exists -- if it does, delete it:
if exist(strcat(var_file_name,'.mat')) == 2
    delete(strcat(var_file_name,'.mat'));
end
%now save all of the appropriate variables
save(var_file_name,char(out_que(1)));
for i = 2:length(out_que)
```

```
   save(var_file_name,char(out_que(i)),'-append');
end
%save the out_que itself so it can be used for re-saving a modified version of the
variables later
save(var_file_name,'out_que','-append');
```

Now that this data is collected and stored to hard-disk, the input file itself can be generated.

Input File Generation

The code is provided here:

```
%gen_frap_input.m

function gen_frap_input(data,input_file_name);


load(data);

fid = fopen(input_file_name,'w');
gen_frap_top(fid); %creates the un-changing beginning to the input deck

for i=1:length(out_que)

    switch char(out_que(i))

        case 'im'
            gen_im(fid,im); %this must be the first in section $FRPCN
        case 'na'
            gen_na(fid,na);
        case 'mechan'
            gen_mechan(fid,mechan);
        case 'ngasr'
            gen_ngasr(fid,ngasr); %this must be the last in section $FRPCN
        case 'cpl'
            gen_cpl(fid,cpl); %this must be the first in section $FRPCON
        case 'crdt'
            gen_crdt(fid,crdt);
        case 'thkcld'
            gen_thkcld(fid,thkcld);
        case 'thkgap'
            gen_thkgap(fid,thkgap);
        case 'dco'
            gen_dco(fid,dco);
        case 'pitch'
            gen_pitch(fid,pitch);
        case 'den'
            gen_den(fid,den);
        case 'dspg'
            gen_dspg(fid,dspg);
        case 'fa'
            gen_fa(fid,fa);
        case 'dspgw'
            gen_dspgw(fid,dspgw);
        case 'enrch'
            gen_enrch(fid,enrch);
        case 'fgpav'
            gen_fgpav(fid,fgpav);
```

```
        case 'hdish'
            gen_hdish(fid,hdish);
        case 'hplt'
            gen_hplt(fid,hplt);
        case 'icm'
            gen_icm(fid,icm);
        case 'gadoln'
            gen_gadoln(fid,gadoln);
        case 'icor'
            gen_icor(fid,icor);
        case 'idxgas'
            gen_idxgas(fid,idxgas);
        case 'iplant'
            gen_iplant(fid,iplant);
        case 'iq'
            gen_iq(fid,iq);
        case 'jdlpr'
            gen_jdlpr(fid,jdlpr);
        case 'totl'
            gen_totl(fid,totl);
        case 'roughc'
            gen_roughc(fid,roughc);
        case 'roughf'
            gen_roughf(fid,roughf);
        case 'vs'
            gen_vs(fid,vs);
        case 'nunits'
            gen_nunits(fid,nunits);
        case 'rsntr'
            gen_rsntr(fid,rsntr);
        case 'nsp'
            gen_nsp(fid,nsp);
        case 'p2'
            gen_p2(fid,p2);
        case 'tw'
            gen_tw(fid,tw);
        case 'go'
            gen_go(fid,go);
        case 'qmpy'
            gen_qmpy(fid,qmpy,im);
        case 'time'
            gen_time(fid,time,im);
        case 'slim'
            gen_slim(fid,slim); %this one must always be last to get the $end
    end
end

st =fclose(fid);
```

The above function works in a way that is analogous to the input-file generation script for the Matlab-VIPRE Interface – only the function names are different. The code for actually writing the input file is provided here:

```
%gen_frap_top.m

function gen_frap_top(fid)

str = 'FILE05=''nullfile'', STATUS=''UNKNOWN'', FORM=''FORMATTED'',';
str2 = '        CARRIAGE CONTROL ='NONE'';
str3 = 'FILE06=''outIrisO'',  STATUS=''UNKNOWN'', CARRIAGE CONTROL = ''LIST''';
str4 = '/*****************************************************************************';
fprintf(fid,'\n%s \n',str);
fprintf(fid,'%s\n',str2);
```

```
fprintf(fid,'%s\n',str3);
fprintf(fid,'%s\n',str4);


%gen_im.m

function gen_im(fid,im)

str = sprintf('im = %4.0f',im);
str1=sprintf('$frpcn');
fprintf(fid,'\n%s',str1);
fprintf(fid,'\n%s,\n',str);


%gen_na(fid,na)

function gen_na(fid,na)

str = sprintf('na=%4.0f',na);
fprintf(fid,'%s,\n',str);


%gen_mechan(fid,mechan)

function gen_mechan(fid,mechan)

str = sprintf('mechan=%4.0f',mechan);
fprintf(fid,'%s,\n',str);


%gen_ngasr(fid,ngasr)

function gen_ngasr(fid,ngasr)

str = sprintf('ngasr=%4.0f',ngasr);

fprintf(fid,'%s,\n',str);
fprintf(fid,'$end\n');


%gen_thkcld(fid,thkcld)

function gen_thkcld(fid,thkcld)

str = sprintf('thkcld=%7.5f',thkcld);

fprintf(fid,'%s,\n',str);

%gen_cpl(fid,cpl)

function gen_cpl(fid,cpl)

str = sprintf('cpl=%5.2f',cpl);
fprintf(fid,'$frpcon\n');
fprintf(fid,'%s,\n',str);

%gen_crdt(fid,crdt)

function gen_crdt(fid,crdt)

str = sprintf('crdt=%5.2f',crdt);

fprintf(fid,'%s,\n',str);

%gen_thkgap(fid,thkgap)

function gen_thkgap(fid,thkgap)
```

```
str = sprintf('thkgap=%7.5f',thkgap);
fprintf(fid,'%s,\n',str);
%gen_dco(fid,dco)
function gen_dco(fid,dco)
str = sprintf('dco=%7.5f',dco);
fprintf(fid,'%s,\n',str);
%gen_pitch(fid,pitch)
function gen_pitch(fid,pitch)
str = sprintf('pitch=%7.5f',pitch);
fprintf(fid,'%s,\n',str);
%gen_den(fid,den)
function gen_den(fid,den)
str = sprintf('den=%5.2f',den);
fprintf(fid,'%s,\n',str);
%gen_dspg(fid,dspg)
function gen_dspg(fid,dspg)
str = sprintf('dspg=%7.5f',dspg);
fprintf(fid,'%s,\n',str);
%gen_fa(fid,fa)
function gen_fa(fid,fa)
str = sprintf('fa=%7.5f',fa);
fprintf(fid,'%s,\n',str);
%gen_dspgw(fid,dspgw)
function gen_dspgw(fid,dspgw)
str = sprintf('dspgw=%7.5f',dspgw);
fprintf(fid,'%s,\n',str);
%gen_enrch(fid,enrch)
function gen_enrch(fid,enrch)
str = sprintf('enrch=%7.5f',enrch);
fprintf(fid,'%s,\n',str);
%gen_fgpav(fid,fgpav)
function gen_fgpav(fid,fgpav)
str = sprintf('fgpav=%7.5f',fgpav);
fprintf(fid,'%s,\n',str);
%gen_hdish(fid,hdish)
```

```
function gen_hdish(fid,hdish)
str = sprintf('hdish=%7.5f',hdish);
fprintf(fid,'%s,\n',str);
%gen_hplt(fid,hplt)
function gen_hplt(fid,hplt)
str = sprintf('hplt=%7.5f',hplt);
fprintf(fid,'%s,\n',str);
%gen_icm(fid,icm)
function gen_icm(fid,icm)
str = sprintf('icm=%7.0f',icm);
fprintf(fid,'%s,\n',str);
%gen_gadoln(fid,gadoln)
function gen_gadoln(fid,gadoln)
str = sprintf('gadoln=%7.5f',gadoln);
fprintf(fid,'%s,\n',str);
%gen_icor(fid,icor)
function gen_icor(fid,icor)
str = sprintf('icor=%7.0f',icor);
fprintf(fid,'%s,\n',str);
%gen_idxgas(fid,idxgas)
function gen_idxgas(fid,idxgas)
str = sprintf('idxgas=%7.0f',idxgas);
fprintf(fid,'%s,\n',str);
%gen_iplant(fid,iplant)
function gen_iplant(fid,iplant)
str = sprintf('iplant=%7.0f',iplant);
fprintf(fid,'%s,\n',str);
%gen_iq(fid,iq)
function gen_iq(fid,iq)
str = sprintf('iq=%7.0f',iq);
fprintf(fid,'%s,\n',str);
%gen_jdlpr(fid,jdlpr)
function gen_jdlpr(fid,jdlpr)
str = sprintf('jdlpr=%7.0f',jdlpr);
fprintf(fid,'%s,\n',str);
%gen_totl(fid,totl)
```

```
function gen_totl(fid,totl)
str = sprintf('totl=%7.3f',totl);
fprintf(fid,'%s,\n',str);
%gen_roughc(fid,roughc)
function gen_roughc(fid,roughc)
str = sprintf('roughc=%10.9f',roughc);
fprintf(fid,'%s,\n',str);
%gen_roughf(fid,roughf)
function gen_roughf(fid,roughf)
str = sprintf('roughf=%10.9f',roughf);
fprintf(fid,'%s,\n',str);
%gen_vs(fid,vs)
function gen_vs(fid,vs)
str = sprintf('vs=%7.2f',vs);
fprintf(fid,'%s,\n',str);
%gen_nunits(fid,nunits)
function gen_nunits(fid,nunits)
str = sprintf('nunits=%5.0f',nunits);
fprintf(fid,'%s,\n',str);
%gen_rsntr(fid,rsntr)
function gen_rsntr(fid,rsntr)
str = sprintf('rsntr=%9.3f',rsntr);
fprintf(fid,'%s,\n',str);
%gen_nsp(fid,nsp)
function gen_nsp(fid,nsp)
str = sprintf('nsp=%9.0f',nsp);
fprintf(fid,'%s,\n',str);
%gen_p2(fid,p2)
function gen_p2(fid,p2)
str = sprintf('p2=%9.3f',p2);
fprintf(fid,'%s,\n',str);
%gen_tw(fid,tw)
function gen_tw(fid,tw)
str = sprintf('tw=%9.3f',tw);
fprintf(fid,'%s,\n',str);
```

```
%gen_go(fid,go)

function gen_go(fid,go)

str = sprintf('go=%10.2f',go);

fprintf(fid,'%s,\n',str);

%gen_qmpy(fid,qmpy)

function gen_qmpy(fid,qmpy,im)

str = strcat('qmpy=',num2str(im),'*',num2str(qmpy));

fprintf(fid,'%s,\n',str);

%gen_time(fid,time,im)

function gen_time(fid,time,im)

fprintf(fid,'time= ');
for i = 1:im
    if(mod(i,7)==0)
        fprintf(fid,'\n');
    end
    fprintf(fid,'%5.0f,',i*time);
end

fprintf(fid,'\n');

%gen_slim(fid,slim)

function gen_slim(fid,slim)

str = sprintf('slim=%5.4f',slim);

fprintf(fid,'%s,\n',str);
fprintf(fid,'\n $end'); %ending
```

FRAPCON Execution Coordination

The FRAPCON execution coordination scripts have a function similar to those for the Matlab-VIPRE interface described in Appendix 2. In general this script is responsible for copying the automatically generated input file to the directory where the FRAPCON executable resides, and copy the output files back to the Matlab working directory where the results can be parsed and analyzed.

The code is presented here:

```
function frap_run(inputfile)
%takes a frapcon inputfile as input, and returns a stusum.txt that is a product of the
%frapcon run. %inputfile is a string
outdirectory = pwd;%strcat(inputfile,'-out');
delete('fuel_data.mat');
current_dir = pwd;
copyfile(inputfile,'c:\matlab_sv13\bin\frapcon\frapcon.txt');
%transfer control to the vipre directory
cd('c:\matlab_sv13\bin\frapcon');
%ensure that I have access to the callVipre.dll
```

```
addpath('c:\matlab_sv13\bin\frapcon');
%execute the system call
callfrapcon;
%now delete the stupid files

delete('frapcon.txt');
delete('nullfile');

%copy the useful files
copyfile('outIrisO',strcat(outdirectory,'\outIrisO'));

copyfile('stusum.txt',strcat(outdirectory,'\stusum.txt'));
copyfile('stustrain.txt',strcat(outdirectory,'\stustrain.txt'));
copyfile('stusum2.txt',strcat(outdirectory,'\stusum2.txt'));
copyfile('stusum3.txt',strcat(outdirectory,'\stusum3.txt'));
delete('stusum.txt');
delete('stustrain.txt');
delete('stusum2.txt');
delete('stusum3.txt');
delete('outIrisO');
%cd(current_dir);
%copyfile('parse_stusum.m',strcat(outdirectory,'\parse_stusum.m'));
%cd current_dir;
cd(outdirectory);
parse_stusum;
%cd .. ;
```

Output Data Parsing

As with the Matlab-VIPRE interface, once the output files are placed in the Matlab working directory, the data must be parsed to be in a format that can be used by the analyst. This task is carried out by the next script:

```
%parse_stusum.m

function parse_stusum
COLUMNS = 22; %number of data columns -- this shouldn't change
NUM_POWER_STEPS = 71; %this could be different for every file
NUM_AXIAL_NODES = 12;

load stusum.txt; %bring the data into the space
load stustrain.txt;
load stusum2.txt;

%initialize data structures
day = stusum(:,2);
peak_node = stusum(:,3);
burnup = stusum(:,4); %MWd\kgU
power = stusum(:,5); %Kw\ft
clad_od_temp = stusum(:,6);
clad_ave_temp = stusum(:,7);
clad_id_temp = stusum(:,8);
gap = stusum(:,9); %mils
fuel_od_temp = stusum(:,10);
fuel_ave_temp = stusum(:,11);
fuel_cl_temp = stusum(:,12);
cont_psi = stusum(:,13);
clad_hoop_stress = stusum(:,14);
clad_axial_stress = stusum(:,15);
clad_strain_pct = stusum(:,16);
fuel_od = stusum(:,17);
gap_cond = stusum(:,18);
```

```
gap_pressure = stusum(:,19);
fission_gas_rel = stusum(:,20);
zirc_oxide_mils = stusum(:,21);
h_2_ppm = stusum(:,22);

%stustrain section.  There are 8 pairs of data items for


fuel_cr = zeros(NUM_POWER_STEPS,1);
fuel_sw=fuel_cr;
fuel_exp=fuel_cr;
fuel_rel=fuel_cr;
clad_perm=fuel_cr;
clad_recov=fuel_cr;
clad_tot=fuel_cr;
index=0;
for i=1:6:NUM_POWER_STEPS
    index = index+1;
    fuel_cr(index)=stustrain(i,1);
    fuel_sw(index)=stustrain(i+1,1);
    fuel_exp(index)=stustrain(i+2,1);
    fuel_rel(index)=stustrain(i+3,1);
    clad_perm(index)=stustrain(i+4,1);
    clad_recov(index)=stustrain(i+5,1);
    clad_tot(index)=stustrain(i+6,1);
end

ox_layer_thick_mic = zeros(NUM_AXIAL_NODES,NUM_POWER_STEPS);

index = 0;

for j = 1:NUM_POWER_STEPS
    for i=1:NUM_AXIAL_NODES
        index=index+1;
        ox_layer_thick_mic(i,j)=stusum2(index,6);
    end
end


clear index;
clear stusum;
clear stustrain;
clear stusum2;
clear stusum3;
save fuel_data;
```

Utility Functions

Initiation of FRAPCON Execution

In exactly the same manner as VIPRE was instantiated in the Matlab-VIPRE Interface, the FRAPCON executable is called for the Matlab-FRAPCON Interface. The code for the required dynamically-linked library is provided here. More complete explanation of the structure of this file is provided in Appendix 2.

```
#include "mex.h"

#include <stdlib.h>
```

```
#include <stdio.h>


void mexFunction(int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[])
{
  printf("\nCalling Frapcon...\n ");
   system("frapstu1");
   return;
}
```

Change of FRAPCON Input

As with the Matlab-VIPRE Interface, the capability to alter a set of input-data is required. This capability is provided with the following function:

```
%change_frap_input.m

function change_frap_input(input_var,field,value)

load(input_var)

expr = strcat(field,'=',num2str(value),';');
%assignin('caller',field,value);
eval(expr);

if exist(strcat(input_var,'.mat')) == 2
    delete(strcat(input_var,'.mat'));
end
%now save all of the appropriate variables
save(input_var,char(out_que(1)));
for i = 2:length(out_que)
   save(input_var,char(out_que(i)),'-append');
end
%save the out_que itself so it can be used for re-saving a modified version of the
variables later
save(input_var,'out_que','-append');
```

Matlab-FRAPCON Interface Example Application

The following is provided as an example of the type of analysis that can be performed with the Matlab-FRAPCON Interface.

For Very Long Cycle Length (VLCL) cores, it is often desireable to add an amount of burnable neutron poison to provide some reactivity control early in core life. One possible option is to use Gadolinium Oxide for this purpose, due to this compound's high neutron absorption cross-section. Unfortunately, use of Gadolinium in the fuel, has the tendancy to reduce the thermal conductivity of the fuel, and thus directly acts to increase the fuel centerline temperature throughout life (for a given power level). This increase in fuel temperature, in turn, results in increased fuel swelling and fission gas release.

To quantify these effects, FRAPCON was used. A reference fuel model was input, and the Gadolinium concentration was arbitrarily varied between 0 and 15 percent, and the results are plotted.[a]

The code is presented here:

```
%gad_vary.m

MIN_GAD_CONC = 0;
MAX_GAD_CONC = 15; %percent
NUM_STEPS = 5;
gad_space = linspace(MIN_GAD_CONC,MAX_GAD_CONC,NUM_STEPS);
num_power_steps = 71; %this is known -- may change
INPUT_FILE_NAME = 'test_inp';
var_file_name = 'test_var';
frap_input_data(var_file_name); %initialize the data
%make sure I have the files available for modification
gen_frap_input(var_file_name,INPUT_FILE_NAME);

max_fuel_temp = zeros(num_power_steps,NUM_STEPS);
max_fuel_od = max_fuel_temp; %initialize the variables

for k=1:NUM_STEPS
    %change the input to the required gadolinium concentration
    change_frap_input(var_file_name,'gadoln',gad_space(k));
    gen_frap_input(var_file_name,INPUT_FILE_NAME);
    frap_run(INPUT_FILE_NAME);
    load fuel_data;

    %data to be saved...
    test_fuel_cl_temp(:,k) = fuel_cl_temp;
    test_fuel_od(:,k) = fuel_od; %both of these are given as a function of burnup
    test_clad_strain(:,k) = clad_strain_pct;
    test_fiss_gas_rel(:,k) = fission_gas_rel;
    test_gap(:,k)=gap;
    test_gap_cond(:,k) = gap_cond;
```

[a] Only the code is provided here. The purpose of this section is to show how to piece together the low-level bits of code presented earlier, into a script that will perform some useful analysis (that would have been more difficult without the script interface).

```
end


%plot the data
plot(burnup,test_fuel_cl_temp(:,1));
hold on
for i=2:NUM_STEPS-1
    plot(burnup,test_fuel_cl_temp(:,i),'-k');
end
plot(burnup,test_fuel_cl_temp(:,NUM_STEPS),'-r');
title('Maximum Fuel Centerline Temperature vs Burnup')
ylabel('Max Fuel CL Temp (F)')
xlabel('Burnup')
hold off
figure

plot(burnup,test_fuel_od(:,1));
hold on
for i=2:NUM_STEPS - 1
    plot(burnup,test_fuel_od(:,i),'-k')
end
plot(burnup,test_fuel_od(:,NUM_STEPS),'-r');
title('Maximum Fuel Outside Diameter vs Burnup')
ylabel('Max Fuel OD (inches)')
xlabel('Burnup')

figure
plot(burnup,test_clad_strain(:,i));
hold on
for i=2:NUM_STEPS-1
    plot(burnup,test_clad_strain(:,i),'-k');
end
plot(burnup,test_clad_strain(:,NUM_STEPS),'-r');
title('Clad Percent Strain vs Burnup')
ylabel('Percent Clad Strain')
xlabel('Burnup')
hold off

figure
plot(burnup,test_fiss_gas_rel(:,i));
hold on
for i=2:NUM_STEPS-1
    plot(burnup,test_fiss_gas_rel(:,i),'-k');
end
plot(burnup,test_fiss_gas_rel(:,NUM_STEPS),'-r');
title('Percent Fission Gas Release vs Burnup')
ylabel('Percent Fission Gas Release')
xlabel('Burnup')
hold off

figure
plot(burnup,test_gap(:,i));
hold on
for i=2:NUM_STEPS-1
    plot(burnup,test_gap(:,i),'-k');
end
plot(burnup,test_gap(:,NUM_STEPS),'-r');
title('Gap Size vs Burnup')
ylabel('Gap Size (mills)')
xlabel('Burnup')
hold off

figure
plot(burnup,test_gap_cond(:,i));
hold on
for i=2:NUM_STEPS-1
    plot(burnup,test_gap_cond(:,i),'-k');
end
plot(burnup,test_gap_cond(:,NUM_STEPS),'-r');
title('Gap Conductance vs Burnup')
ylabel('Gap Conductance (W/m^2-K)')
```

```
xlabel('Burnup')
hold off
```

Modifications to FRAPCON

Some minor modifications to the FRAPCON source code were required to allow for the script-based interface. These modifications are all centered around the requirement to organize key output data in a format that is convenient both for transfer into the Matlab workspace context, and for parsing the numeric data for subsequent analysis.

As with the Matlab-VIPRE interface, the most convenient form for the output data, from the perspective of the scripted interface, is for all of the key data to exist in tabular form in a ASCII text-file. The key properties of this file are that:

- There are only numbers, no text
- The numbers are arranged into rows. Each row must have the same number of entries.[b]

With the output files in this format, the data can be loaded into the Matlab workspace with a single command, and the data can be parsed in a relatively straight-forward manner.

The second requirement listed above dictated that there must be four additional output files produced. The idea, during the development of this interface was that: there are certain tables of particular interest in the output file. A single output file would be dedicated to capturing the output that would otherwise go to the particular table of interest. If there is another table that is interesting, an additional output file would be established as the repository for this data. This methodology also simplified the logic required for the subsequent data parsing.[c]

The additional files created are named:

- Stusum.txt

[b] E.g. It is not permissible that the first row have 6 entries, while the second row has only 5.
[c] Only one table format would have to be extracted from the data file. If the table is printed, for example, every power-step, the number of power-steps is a known quantity, therefore the table was printed in the output file a known number of times. This gives the programmer complete knowledge of the structure of a given output file based on a simple parameter – for this example, the number of power-steps. If there were more than one table printed to the same input file, the tables may not have been printed in an entirely predictable fashion. This would complicate the parsing-routine logic, and is thus undesirable.

- Stusum2.txt
- Stusum3.txt
- Stustrain.txt

They are created with the following code, added to the source file: iofiles.f:

```
open(unit=7,file='stusum.txt',status='new',form='formatted')
open(unit=8,file='stustrain.txt',status='new',form='formatted')
open(unit=9,file='stusum2.txt',status='new',form='formatted')
open(unit=10,file='stusum3.txt',status='new',form='formatted')
```

In the source file print2.f, the following changes were then required to capture the output:

```
if (nunits.eq.1) write (7,990) iit,aaa(1),jpeak1
     ,(aaa(i),i=2,7),(aaa(i),i=9,18),aaa(8),aaa(19),aaa(20)
if (nunits.eq.0) write (7,1000) iit,aaa(1),jpeak1
     ,(aaa(i),i=2,7),(aaa(i),i=9,18),aaa(8),aaa(19),aaa(20)
990   format (i3,1x,f6.1,1x,i2,2x,f7.2,f6.2,1x,3(f7.0),f5.2,1x
    +,3(f7.0),1x,f6.0,1x,f7.0,1x,f10.0,f10.4,f10.5,1x,f7.0,1x,f5.0,1x,f5.1
    +,f6.2,1x,f6.1)
1000  format (i3,1x,f6.1,1x,i2,2x,f7.2,f6.2,1x,3(f5.0),f5.2,1x
    +,3(f5.0),1x,f6.0,1x,f7.3,1x,f7.3,f7.4,1x,f8.6,1x,f7.0,f6.3,1x,f5.1
    +,f6.2,1x,f6.1)

!following lines added to extract output data for stustrain.txt
          write(8,1100) denrml,denrmc
1100      format(f11.5,2x,f11.5)
          write(8,1110) swlrml, swlrmc
1110  format(f11.5,2x,f11.5)
          write(8,1120) exprml,exprmc
1120  format(f11.5,2x,f11.5)
          write(8,1130) relocm,relocs
1130  format(f11.5,2x,f11.5)
!         write(8,1140) rco, rcom
1140  format(f11.5,2x,f11.5)
          write(8,1150) creapl, creapm
1150  format(f11.5,2x,f11.5)
          write(8,1160) expnrl, expnrm
1160      format(f11.5,2x, f11.5)
          write(8,1170) totcrl, totcrm
1170      format(f11.5,2x f11.5)
!=============================================================

write(9,1910)im1,a(lcn2h2+im1-1),a(lcexh2+im1-1),epsunp,oxmils
    +   ,oxmicr, a(lfluen+im1), fmgrpct
```

```
1910     format(i3,6x,f8.2,8x,f8.2,6x,f6.3,9x,f6.4,
   +   1x,f7.3, 7x, e10.4, 7x, f6.2)

write (10,891) im1,tfrk,a(ltmpds+ij),tbark,a(ltbar+im1),tcak,a(
   +   ltca+im1),tblkk,tblkf,cpdltx,eppra,epphpi,epphpo,eppax,epgro
891   format (i2,3x,2(f7.1,2x,f7.1,2x),2x,f5.0,2x,f5.0,3x,f
   +5.0,2x,f5.0,2x,1pe8.2,8x,0pf5.2,2x,f5.2,3x,f5.2,2x,f5.2,
   +4x,f5.2)
```

Appendix 8 RELAP Interface

The Matlab-VIPRE-RELAP Interface is not finished, but some of the key tools are put in place. A schmatic view of the interface is provided in Figure 44. The basic form of the Matlab-VIPRE Interface is left intact. The difference is that boundary-condition data for transient analysis is passed through the file system from RELAP to VIPRE. As with the Matlab-VIPRE Interface, all of the movement of data and coordination of operations takes place within the Matlab environment.

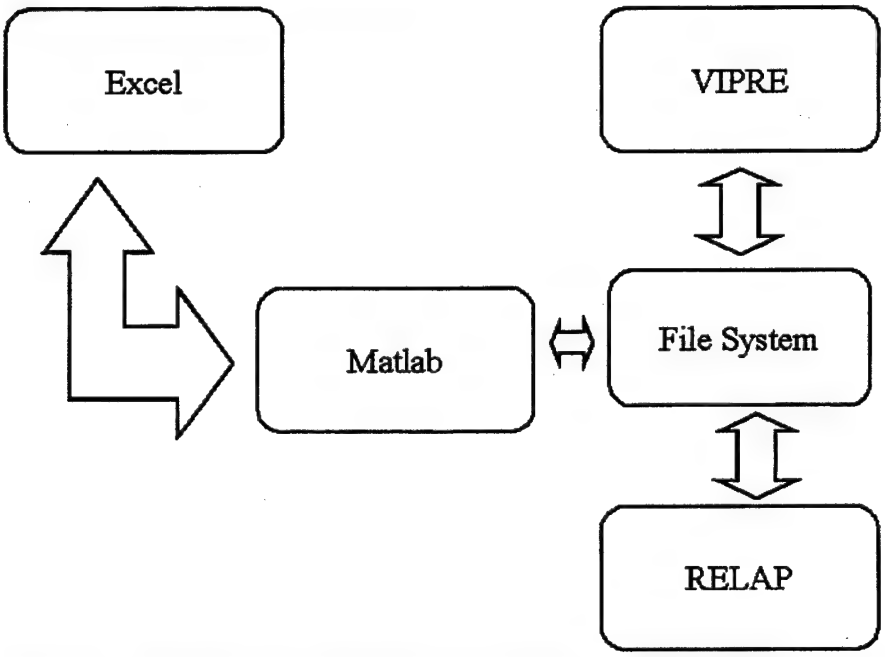


Figure 44: Schematic representation of the Matlav-VIPRE-RELAP Interface.

The interface, to the extent that it is functioning at the time of this writing, has as a prerequisite a valid RELAP input deck for the particular transient that is of interest. Modifications have been made to the RELAP source code such that a separate output file is produced that has, without any other adornments, the 'minor edits' as specified in the input deck.

The two main methods of output from the RELAP program is through so-called 'Major Edits' and 'Minor Edits'. The major edit is, put briefly, a 'dump' of the status of

every component of the RELAP model. This is a large body of information that, while presenting a great deal of data to the user, is difficult to use due to the vast quantity of information displayed. The major edits are output at intervals (as determined by the number of discrete time steps taken within the simulation) set by the user in the input deck.

The minor edits are specific bits of data that are requested at intervals that are generally shorter than the major edits. For the simulation, there may be particular quantities that the analyst may like to keep a close eye on – for example, the core inlet mass flow rate. The user would direct this value to be output as a minor edit – which is generally a much smaller quantity of data than what is provided with the major edit, and allows the desired information to be found within the output file more quickly and, provided the minor edits are output at shorter intervals than the major edits, the data among the minor edits will have sharper temporal resolution.

The portion of the source code that writes the minor edits to the usual output file has been modified so as to send this same data (though without data labels, or textual information of any kind – only numbers are allowed) to an ordinary ASCII text file that can easily be loaded and parsed by a relatively simple Matlab Script. The modified portion of the RELAP source code is given here:

```
*define win32dvf
*define erf
*define fourbyt
*define hconden
*define impnon
*define in32
*define newnrc
*define ploc
*define sphaccm
*define unix
*define noselap
*define noextvol
*define noextv20
*define noextsys
*define noextjun
*define noextj20
*define noparcs
*define nonpa
*define nomap
*define logp
*deck mirec
      subroutine mirec
c  $Id: mirec.ff,v 1.1 2001/02/01 23:17:28 r5qa Exp dbarber $
c
c  mirec transfers results of time step to save area for minor edits and
```

```
c edits the information if save area is full.
c
c Cognizant engineer: rjw.
c
      implicit none
      include 'comctl.h'
      include 'contrl.h'
      include 'fast.h'
      include 'miedtc.h'
      include 'ufiles.h'
c
c Local variables.
      integer i,ix,j,l1,l1a,l2,l2x,l3,l3x,l4,m,n,n1

            integer istu
            istu=69
c
c Get packed quantities to control store of results.
      l1 = filndx(16)
      m = mipck(2,l1 + 4)
      n = mipck(2,l1)
      l3 = mipck(2,l1 + 3)
      if (done) 21,22,23
   23 if (m .ne. 0) go to 16
      return
   21 done = -done
   22 l1a = mipck(2,l1+1)
      l2 = l3 + m*n
c Store results into save area.
      do 15 i = 1,n
   11   mihold(l2) = fa(micode(2,l1+1))
   12   if (unito) go to 14
        if (miconv(l1) .lt. 0.0d0) go to 13
        mihold(l2) = mihold(l2)*miconv(l1)
        go to 14
   13   mihold(l2) = (mihold(l2)+miconv(l1))*1.8d0
   14   l1 = l1 + l1a
        l2 = l2 + 1
   15 continue
c Increment number of edits and test if save area is full.
      m = m + 1
      if (m .ne. 50) go to 100
c Edit accumulated minor edit data.
   16 l1a = 9
      l2 = l3
      n1 = n - 1
      l4 = mipck(2,filndx(16) + 2)
   19   if (n1 .lt. 9) l1a = n1
        ix = l4 + (l1a-1)*8
      write (output,2001) (milabl(j),milabl(j+1),j=l4,ix,8)


 2001 format('1 time',8x,9(a8,a5))
      write (output,2003) (milabl(j+2),milabl(j+3),j=l4,ix,8)

 2003 format('  (sec)',7x,9(a8,a5))
      write (output,2004) (milabl(j+4),milabl(j+5),j=l4,ix,8)

 2004 format (15x,9(a8,a5))
      write (output,2004) (milabl(j+6),milabl(j+7),j=l4,ix,8)

        l2x = l2
        l3x = l3
        do 20 i = 1,m
          ix = l3x + l1a - 1
          write (output,2002) mihold(l2x),(mihold(j+1),j=l3x,ix)
```

Extract minor edit data with this line

```
            write(istu,2002) mihold(l2x),(mihold(j+1),j=l3x,ix)
 2002  format(1p,g14.6,9g13.5)
       l2x = l2x + n
       l3x = l3x + n
  20   continue
       l3 = l3 + 9
       l4 = l4 + 72
       n1 = n1 - 9
       if (n1 .gt. 0) go to 19
      m = 0
c
  100  mipck(2,filndx(16)+4) = m
      return
      end
c*********************************************************************
c
c Data dictionary for local variables
c
c Number of local variables =  13
c
c i=integer r=real l=logical c=character
c*********************************************************************
c Type   Name                  Definition
c---------------------------------------------------------------------
c i    i                    =
c i    ix                   =
c i    j                    =
c i    l1                   =
c i    l1a                  =
c i    l2                   =
c i    l2x                  =
c i    l3                   =
c i    l3x                  =
c i    l4                   =
c i    m                    =
c i    n                    =
c i    n1                   =
c*********************************************************************
```

Note that the file with unit designation '69' is opened in a different portion of the source code and represents the text file "stuout.txt". One should also note that, with the modification implemented in this way, if more than nine variables are requested to be output with the minor edits (or as would be said: "…if more than nine minor edits are requested…") the modification will fail. The code will output the minor edits using a maximum of nine columns. That is, if more than nine minor edits are requested, there will be two separate 'blocks' of minor edit data printed to the output file (and thus to 'stuout.txt'). This is a problem because the Matlab code used to parse the results stored in 'stuout.txt' requires:

1. That each row have the same number of colums – If there are more than nine minor edit requests, this would only be true if the number of minor edit requests is an integer multiple of nine. Failure of this condition will cause Matlab to report an error.[a]

2. The parsing script assumes that each row of numbers corresponds to the complete set of minor edits for a given time step. If there is an integer multiple of nine minor edits, and thus Matlab is able to load the contents of the file into the workspace context, there would still be a logical error in the parsing script in that only a fraction of the data would be parsed, and only the first block of data would be parsed correctly.

Note that the second requirement could be eliminated by making changes to the data parsing code. It is expected that as the needs of the analysts change, this parsing script will also be changed.

This is the current state of the RELAP interface. Once a satisfactory RELAP input deck is created and modified to output the appropriate minor edits for input to VIPRE (core power, system pressure, core inlet flow rate, core inlet coolant temperature and transient time) it will be left to modify the input data extraction scripts from the Matlab-VIPRE Interface for transient input (i.e. get_excel_input_trans.m – or one of the variants therof such as get_excel_input_trans_w31.m) so that the transient forcing function data is extracted from the RELAP ouput file stuout.txt, rather from a sheet within the Excel workbook as is currently done.

[a] When using the 'load(*filename*)' within the Matlab environment, Matlab requires that each row of numbers contains the same number of columns of data.

Appendix 9 IRIS Open Core Fuel Cycle Analysis

Introduction

The economic attractivness of a given design is, with the exception of safety, probably the most important discriminator between different designs. In this appendix, an economic evaluation of the fuel cycle cost is conducted for the IRIS Open core. This fuel cycle cost analysis takes into consideration the following cost components:

- Uranium purchase, conversion and enrichment costs;
- Fuel fabrication cost;
- Replacement Energy Cost during (planned and unplanned) plant shutdowns; and
- An estimate of O&M costs during planned plant shutdowns.

IRIS Open Model

To perform an economic analysis of the IRIS fuel cycle, several core design inputs are required. These inputs are summarized in Table 19.

IRIS Open Core Design Inputs	
Thermal Power	1000 MWth
Electric Power	335 MWe
Fuel Loading	48.5 Metric Tons Uranium
Fuel Enrichment	4.95 %
Single Batch Discharge Burnup	38,000 MWd/Ton Uranium

Table 19: IRIS Core Design Parameters.

To allow for fuel management strategies other than the single batch "straight burn", the Linear Reactivity Model was applied to determine discharge burnup for N batches. The discharge burnup is then estimated by the following equation:

Equation 27

$$BU_{discharge} = BU_{1-batch} \cdot \frac{2N}{(N+1)}$$

The following assumptions were made regarding plant operation and maintenance schedules:

IRIS Operation and Maintenance Parameters	
Refueling Outage Length	15 Days
Forced Outage Rate	2 %

Table 20: IRIS O&M Parameters.

Economic Model

The methodology applied for the fuel cycle calculations is based on the levelized, lifetime cost method in accordance with OECD/NEA recommendations.[49],[50] All front-end fuel cycle costs and future revenues are referred to the beginning of irradiation. The value of these front-end costs and future revenues are calculated assuming:

- Front-end costs occur prior to the beginning of irradiation according to the schedule given in Table 21.

Front-end Transaction Lead Times	
Transaction	**Lead Time (months)**
Uranium Purchase	24
Uranium Conversion	18
Uranium Enrichment	18
Uranium Fabrication	12

Table 21: Fuel Cycle Front-end Purchase Lead Time.

- The unit cost for the front-end expenditures are listed in Table 22.

Front-end Unit Prices[b]	
Uranium Purchase	27.7 $/kgU
Uranium Conversion	4.9 $/kgU
Uranium Enrichment	108 $/kgSWU
Uranium Fabrication	200 $/kgU

Table 22: Unit Prices for Front-End Transactions.

- Remaining Parameters relevant to the fuel front-end are listed in Table 23.

[b] Prices quoted were obtained at: http://www.uxc.com/review/uxc_prices.html -- a page from the Ux Consulting Group LLC website.

Additional Parameters	
Uranium Feed Enrichment	0.71 %
Tails Enrichment	0.3 %
Discount Rate	8 %

Table 23: Front-end Cost Parameters.

These assumed parameters were then applied to the following economic model:

Front-end Cost Calculation

The front-end fuel cycle costs of Uranium Purchase, Uranium Conversion, Uranium Enrichment and Fuel Fabrication are computed individually and compounded linearly to determine their worth at the beginning of fuel irradiation.

Uranium Cost

The natural Uranium required as an input to the nuclear fuel manufacturing process is computed as a cost per kilogram according to the following equation:

Equation 28

$$\frac{\$_{start\ of\ irradiation}}{kg_{fuel\ product}} = \frac{kg_{feed}}{kg_{fuel\ product}} \cdot \frac{\$}{kg_{feed}} \cdot \left(1 + x\Delta T_{uranium\ purchase}\right)$$

Where: x is the discount rate
ΔT is the required lead time

The quantity of feed required per quantity of fuel product is found as the following function of fuel enrichment, feed enrichment, and tails enrichment:

Equation 29

$$\frac{F}{P} = \frac{x_p - x_t}{x_f - x_t}$$

Where:

$\frac{F}{P}$ = mass of feed per unit product
x_p = fuel (product) weight fraction
x_f = feed weight fraction
x_t = tails weight fraction

Fuel Conversion Cost

The fuel conversion cost is computed as follows:

Equation 30

$$\frac{\$_{start\ of\ irradiation}}{kg_{fuel\ product}} = \frac{kg_{feed}}{kg_{fuel\ product}} \cdot \frac{\$_{conversion}}{kg_{feed}} \cdot \left(1 + x\Delta T_{uranium\ conversion}\right)$$

Fuel Enrichment Cost

The enrichment costs are computed as follows:

Equation 31

$$\frac{\$_{start\ of\ irradiation}}{kg_{fuel\ product}} = SWU \cdot \frac{\$}{SWU} \cdot \left(1 + x\Delta T_{fuel\ enrichment}\right)$$

Here, the amount of SWU (separative work units) required is computed as follows:

Equation 32

$$SWU = \frac{kg_{swu}}{kg_{fuel}} = [V(p) - V(t)] - \frac{x_p - x_t}{x_f - x_t} \cdot [V(f) - V(t)]$$

Where:

$$V(p) = (2x_p - 1)\ln\frac{x_p}{1 - x_p}$$

$$V(f) = (2x_f - 1)\ln\frac{x_f}{1 - x_f}$$

$$V(t) = (2x_t - 1)\ln\frac{x_t}{1 - x_t}$$

Fuel Fabrication Costs

The fuel fabrication costs are computed as follows:

Equation 33

$$\frac{\$_{start\ of\ irradiation}}{kg_{fuel\ product}} = \frac{\$_{fabrication}}{kg_{fuel}} \cdot \left(1 + x\Delta T_{fuel\ fabrication}\right)$$

Total Fuel Front-end Costs

The total fuel front-end cost is the sum of the costs for the mined uranium, uranium conversion, enrichment and fuel fabrication:

Equation 34

$$I_o = \textit{Uranium Purchase Cost} + \textit{Conversion Cost} + \textit{Enrichment Cost} + \textit{Fabrication Cost}$$

Levelized Fuel Cycle Cost

The present worth of the continuous revenue stream from sale of electricity is:

Equation 35

$$E_o = \frac{8766}{1000}\int_o^T lfcc \cdot p \cdot L \cdot \eta \cdot e^{-xt}\,dt \quad \left[\frac{\$}{kg_{HM}}\right]$$

Where:

$$\frac{8766}{1000} = \frac{hr/year}{mills/\$}$$

lfcc = levelized fuel cycle cost $\left[\frac{mills}{kW_e}\right]$

T = core residence time of a fuel assembly (years)

p = specific power $\left[\frac{kW}{kg_{HM}}\right]$

L = capacity factor (%)

η = thermodynamic efficiency $\left[\frac{kW_{th}}{kW_e}\right]$

Formally Equation 35 is solved to equal:

Equation 36

$$E_o = 8.766 \cdot lfcc \cdot p \cdot L \cdot \eta \cdot \left[\frac{1-e^{-xT}}{x}\right]$$

To find the value for levelized fuel cycle cost, Equation 36 is equated to the value of the front-end costs at the beginning of irradiation (Equation 34). The levelized fuel cycle

cost is the cost at which the front-end costs, and the continuous revenue stream from electrical generation are made equal:

Equation 37

$$lfcc = \left[\frac{I_o}{8.766 \cdot p \cdot L \cdot \eta}\right] \cdot \left[\frac{x}{1 - e^{-xT}}\right]$$

Since the fuel discharge burnup can be computed as:

Equation 38

$$Bu = \frac{365 \ days/year}{1000 \ kW_{th}/MW_{th}} \cdot p \cdot L \cdot T$$

Equation 37 can be equivalently expressed in the somewhat more convenient form:

Equation 39

$$lfcc = \left[\frac{I_o}{8.766 \cdot Bu \cdot \eta}\right] \cdot \left[\frac{xT}{1 - e^{-xT}}\right]$$

To perform the actual computation, an expression is also required for the specific power (p) and capacity factor (L).

From a complete core design, the specific power can be computed simply as the ratio of the core themal power (in kW_{th}) to the heavy metal loading (in kilograms).[c] The capacity factor is expressed as follows:

Equation 40

$$L = L'\left[1 - \frac{T_R}{T_C}\right]$$

Where:

L' = Plant availability = 1 – Forced Shutdown Rate
T_R = Length of Refueling Outage (days)

[c] This actual computation can be complicated to some degree by the requirement to account for the fabricated fuel density (as compared to fuel theoretical density) and any fuel pellet design features such as 'dishing'. A general expression for the mass of fuel present as a function of a few key core parameters is thus forgone.

T_C = Length of Refueling Cycle (operating and refueling days)

For the purposes of this analysis, the plant availability is an assumed parameter as is the refueling outage length. The length of the refueling cycle is dependent on the duration of the outage (assumed) and the computed length of operation for a given batch within a fuel cycle. The length of operation of a given batch within a cycle is determined as a function of the specific power, the forced outage rate and the burnup for a given batch. The specific power is computed as described above; the forced outage rate is assumed. The discharge burnup for a given fuel management strategy (i.e. number of batches) is computed using the linear reactivity model in Equation 27. The burnup for a given batch within the fuel cycle is then found by:

Equation 41

$$Bu_{per\,cycle} = \frac{Bu_{discharge}}{\#\,of\ Batches}\left[\frac{MWd}{MT}\right]$$

Equation 42

$$Bu_{per\,day} = p \cdot L' = \frac{1000}{48.5}\cdot(1-.02) = 20.6\left[\frac{MW}{MT}\right]^{d}$$

Therefore

Equation 43

$$T_C = \frac{Bu_{per\,cycle}}{Bu_{per\,day}} + T_R \ \ [days]$$

Thus, the capacity factor can be found; which was the last link in the chain required to determine the levelized fuel cycle cost.

Analysis – Front End Costs

Considering only the parts of the model considered so far, the levelized fuel cycle cost is computed, for a variety of fuel management strategies. In principle, any number of batches may be planned for a given fuel cycle. The number of fuel assemblies to be removed for each batch is simply:

[d] Some actual values given for the IRIS design considered here.

Equation 44

$$\#\text{ Assemblies to Replace Each Refueling} = \frac{\#\ Assemblies\ In\ Core}{\#\ Batches}$$

As a practical matter, an integer number of fuel assemblies must be replaced with each batch at the very least. More practically still, the number of fuel assemblies to be replaced should be an integer multiple of 4. The fuel management strategies considered in this study are given in Table 24.

# of Batches	# Fresh Fuel Assemblies Replaced At Each Refueling
1.00	89
2.02	44
2.47	36
3.18	28
3.71	24
4.45	20

Table 24: Fuel Management Strategies Considered for IRIS.

For these strategies, the following parameters are computed:

# Batches	Discharge Bu (MWd/tU)	Bu/cycle (MWd/tU)	Days/cycle
1.00	38000	38000	1881.15
2.02	50857	25143	1244.67
2.47	54112	21888	1083.54
3.18	57812	18188	900.38
3.71	59858	16142	799.07
4.45	62055	13945	690.33

Table 25: Fuel Management Strategy Parameters.

Considering only front-end costs, the levelized fuel cycle cost for the fuel management strategies presented is given in Figure 45. The parameters used in this analysis are summarized below:

Parameter	Value
Thermal Power	1000 MWth
Electric Power	335 MWe
Fuel Loading	48.5 Metric Tons Uranium
Fuel Enrichment	4.95 %
Uranium Purchase Lead Time (months)	24
Uranium Conversion Lead Time (months)	18
Uranium Enrichment Lead Time (months)	18
Uranium Fabrication Lead Time (months)	12
Uranium Purchase Unit Price	27.7 $/kgU
Uranium Conversion Unit Price	4.9 $/kgU
Uranium Enrichment Unit Price	108 $/kgSWU
Uranium Fabrication Unit Price	200 $/kgU
Uranium Feed Enrichment	0.71 %
Tails Enrichment	0.3 %
Discount Rate	8 %
Forced Outage Rate	2 %

Table 26: Summary of Parameters for Initial Fuel Cycle Cost Analysis.

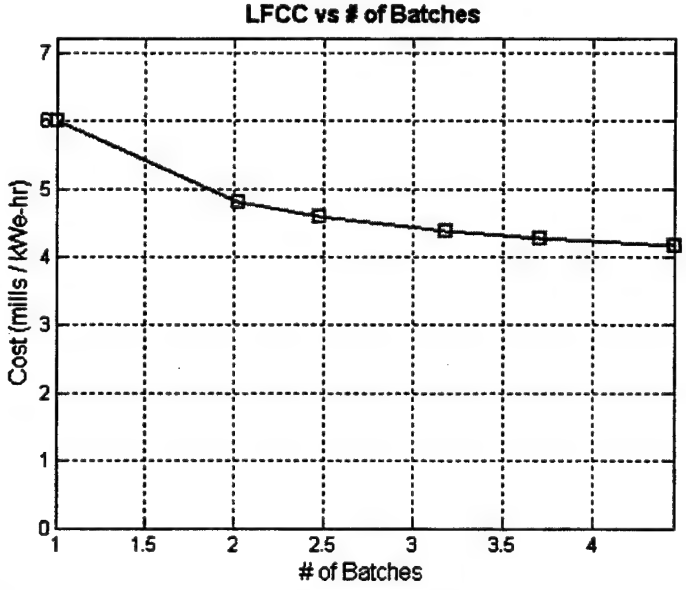


Figure 45: Levelized Fuel Cycle Cost - Considering only Front-End Costs.

The above figure clearly shows the benefit to be expected to fuel cycle costs by increasing the number of batches. As shown in Table 25, the discharge burnup increases with an increasing number of batches. Therefore, by increasing the number of batches, more power is extracted from every kilogram of fuel, thereby increasing the length of the

revenue stream from that mass of fuel, thereby reducing cost. The above analysis, however, neglects the consideration of the costs associated with each shutdown.

Analysis – Total Fuel Cycle

With each plant shutdown a myriad of events must take place. Aside from refueling the core, hundreds of maintenance activities that either require plant shutdown, or were deferred until the next plant shutdown must be accomplished. Most of these considerations are beyond the scope of this analysis. However, an attempt will be made to capture some of the features of these refueling shutdowns and incorporate them into the model to provide some useful information regarding overall costs during a given fuel cycle.

For this model, replacement energy costs will be considered explicitly, as will an estimated shut-down cost. Additionally, some approximations will be made to capture the variability of shutdown duration.

The replacement energy costs are modeled as a flat fee per unit energy. For this analysis, the assumed value is: 30 mills/kWhr. This value is then normalized so as to be given in terms of mills per kWhr that the power plant itself produces in the following way:

Equation 45

$$Replacement\ Energy\ Cost = \frac{e_r P(1-L)T}{PT} = e_r(1-L) \quad \left[\frac{mills}{kWhr}\right]$$

Where:

e_r = marginal cost of replacement energy (mills/kWhr)
P = electrical output of power plant (MW)

Note, that conversion factors to convert the units of electrical power from MW to kW and the fuel residence time from years to days in both the numerator and denominator cancel out.

Additionally, it will be assumed that the cost of every shutdown can be modeled as a linear function of shutdown duration. For this model, the relationship is given as:

Equation 46

Outage Cost = $400 Thousand per day

This value is in addition tot her replacement energy cost, and has been derived from the corresponding cost for current loop PWRs. For a 1,000 MWe PWR, this cost widely varies and may amount to some $10-25 Million per outage. The assumed $400K/day and 15 day outage, leading to a total of $6 Million for IRIS, is larger than the value scaled down by IRIS installed power (about one quarter of the power of a 4-loop PWR), but it was taken as a conservative estimate pending a detailed analysis of the IRIS outage cost.

In a way similar to that of the replacement energy costs, the outage costs are normalized so as to be comparable with the other fuel cycle cost components. This is done in accordance with the following equations:

Equation 47

(Cost-per-shutdown)/(kWhr per batch)

The Cost per shutdown is computed using Equation 46. Equation 45 requires the length of the shutdown in days which is computed using Equation 48. Equation 48 requires one to know how many days of operation occurred between refuelings which is computed using Equation 41 and Equation 42.

As an additional measure, the fuel disposal cost of 1 mill/kWhr is added in order to make a more full accounting of all fuel cycle related costs.

The parameters used in this more complete analysis is provided in Table 27 and the results are displayed in Figure 46.

Parmeter	Value
Thermal Power	1000 MWth
Electric Power	335 MWe
Fuel Loading	48.5 Metric Tons Uranium
Fuel Enrichment	4.95 %
Uranium Purchase Lead Time (months)	24
Uranium Conversion Lead Time (months)	18
Uranium Enrichment Lead Time (months)	18
Uranium Fabrication Lead Time (months)	12
Uranium Purchase Unit Price	27.7 $/kgU
Uranium Conversion Unit Price	4.9 $/kgU
Uranium Enrichment Unit Price	108 $/kgSWU
Uranium Fabrication Unit Price	200 $/kgU
Uranium Feed Enrichment	0.71 %
Tails Enrichment	0.3 %
Discount Rate	8 %
Forced Outage Rate	2 %
Refueling Outage Length	15 Days
Shutdown O&M Cost	$400 K/Day
Replacement Energy Cost	30.0 mills/kWhr
Spent Fuel Disposal Fee	1 mills/kWhr

Table 27: IRIS Parameters for Refined Fuel Cycle Cost Analysis

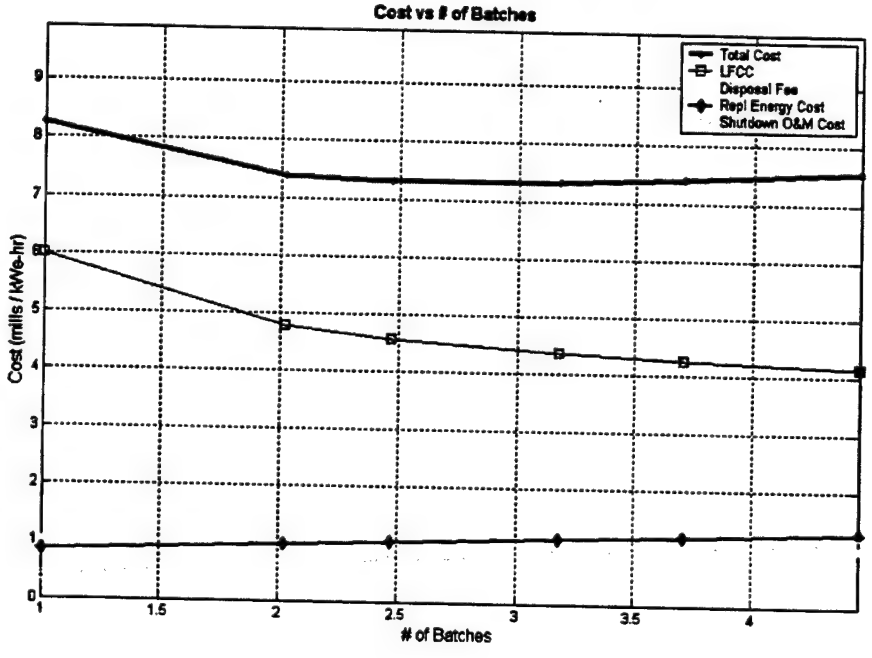


Figure 46: Fuel Cycle Costs for IRIS Including Shutdown O&M, Replacement Energy Costs, and Spent Fuel Disposal Fee.

As a further refinement, the refueling outage length is given as a linear function of the duration between refuelings. This function is:

Equation 48

$$Outage\ Length = 10\,days + 2\,\frac{days}{year\ of\ operation}$$

This is means that for fuel management strategies that call for less frequent shut-downs (i.e. fewer batches), the shutdowns will be longer to allow reloading a larger number of fuel assemblies, as well as accomplishing a greater number of deferred maintenance tasks. The results of this analysis are given in Figure 47.

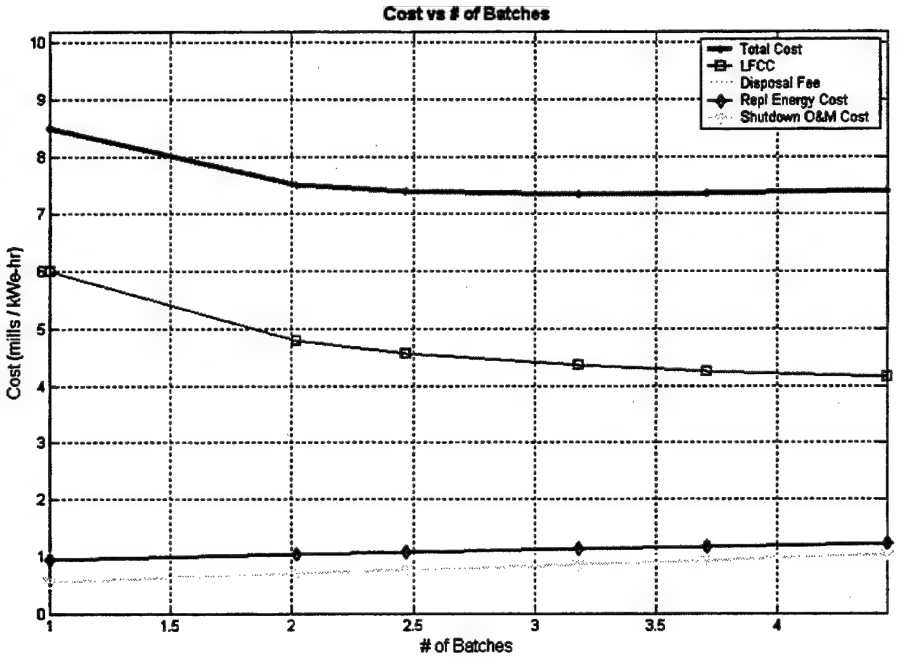


Figure 47: IRIS Fuel Cycle Cost Results with Variable Refueling Outage Length.

Sensitivity Study

Clearly, all of these analyses involve uncertainty with regards to most (if not all) of the parameters. It is useful for the engineer to be aware of which of the assumed parameters influence the outcome of the fuel cycle analysis the most. For these

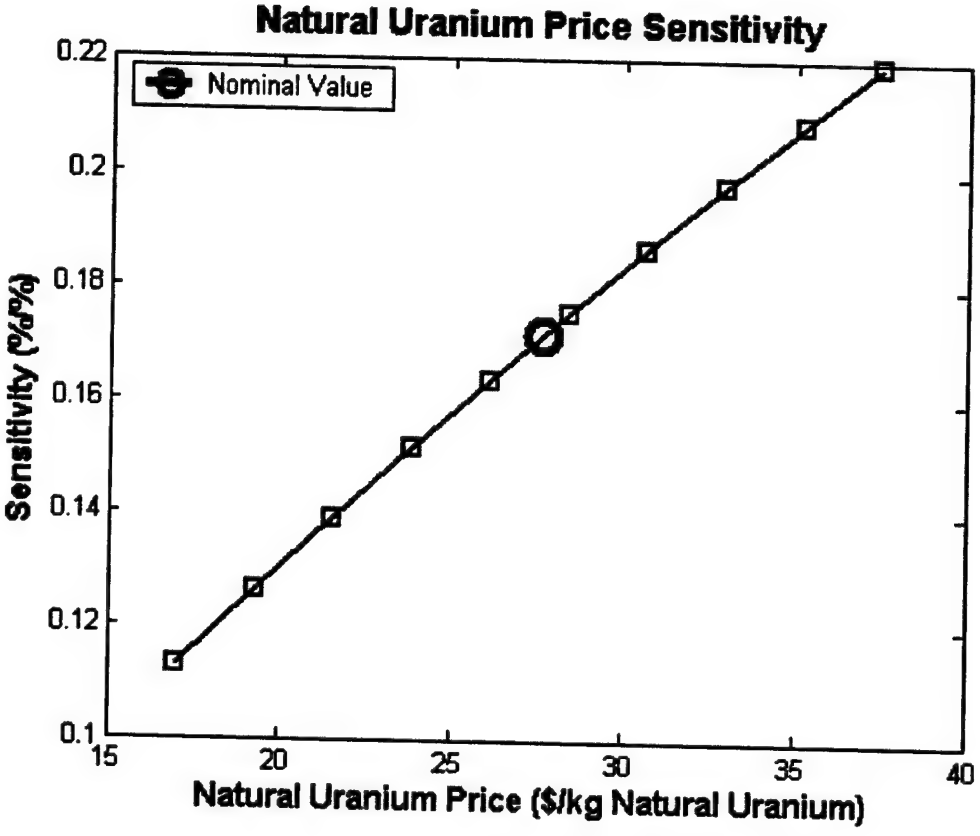
sensitivity results, each parameter discussed is varied individually. The sensitivity is defined as: the percentage change of the *total* cost[e] for every percentage change in the given parameter.

Equation 49

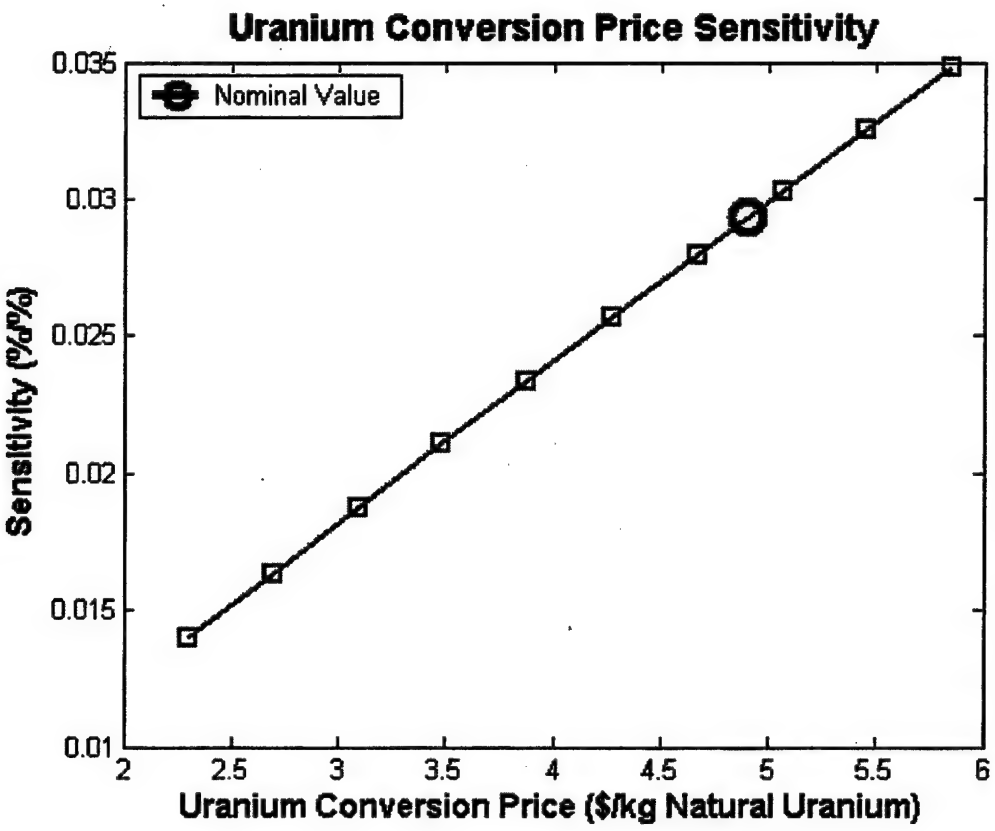
$$\frac{e - e_{ref}}{e_{ref}} \Big/ \frac{c_i - c_{ref}}{c_{ref}}$$

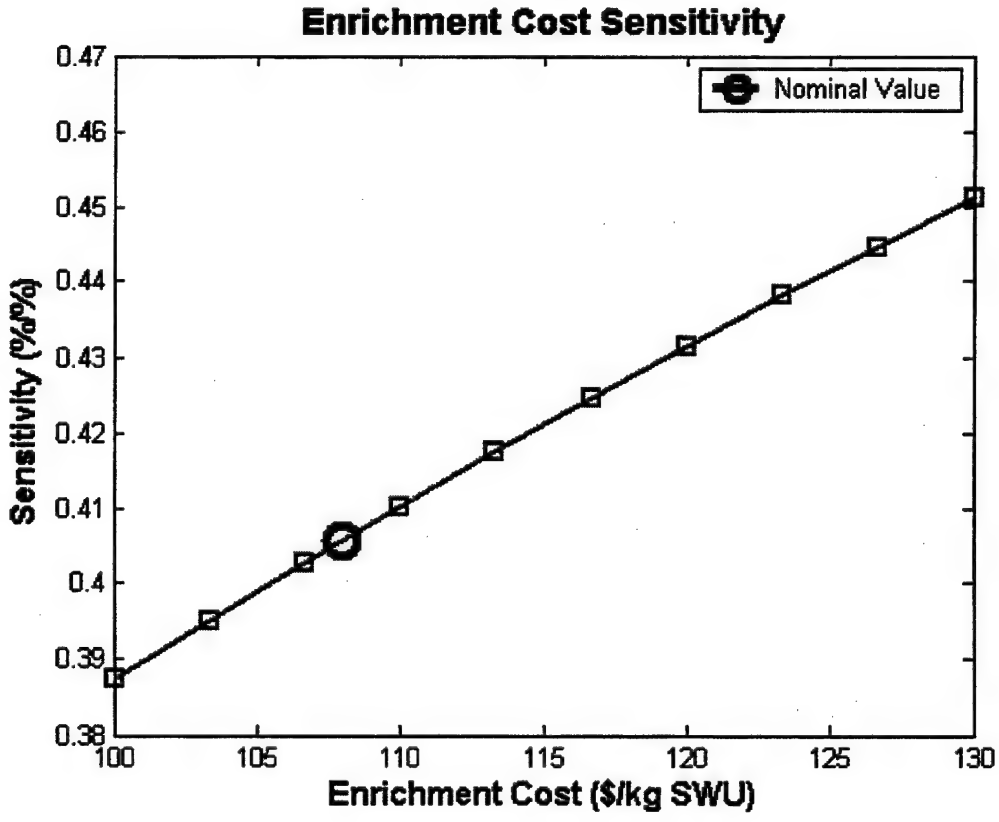
Where:

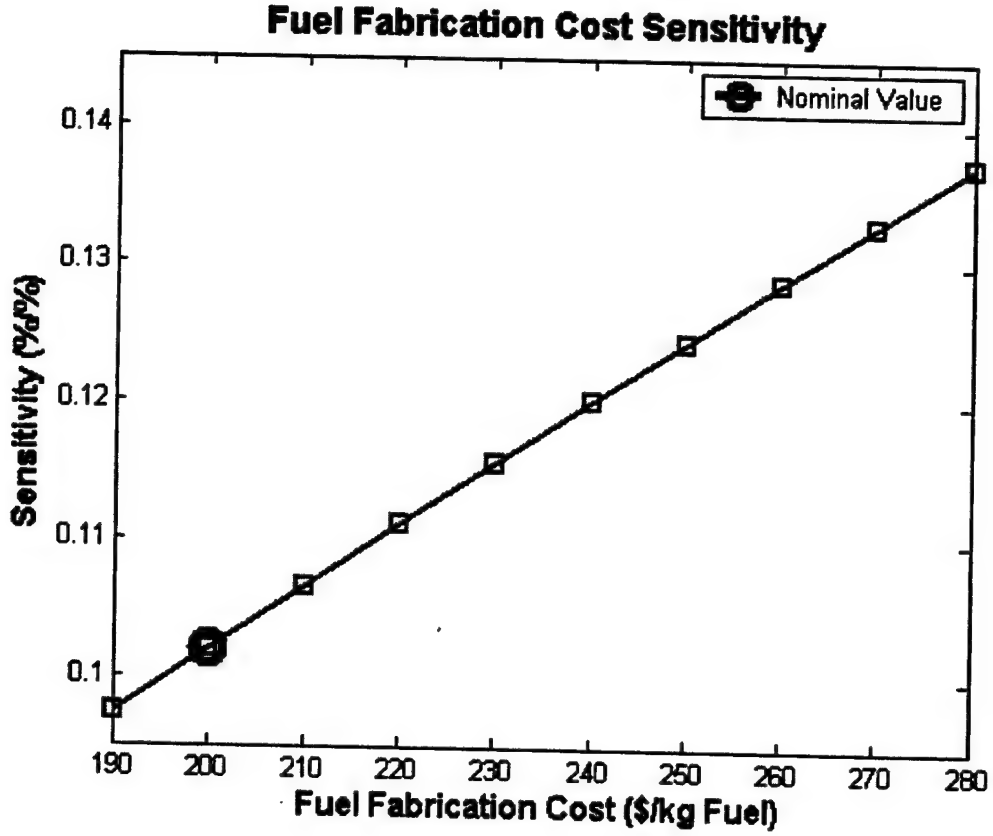
e = total cost with one perturbed parameter
e_{ref} = reference total cost.[f]
c_i = perturbed value of one parameter
c_{ref} = reference value of the parameter.

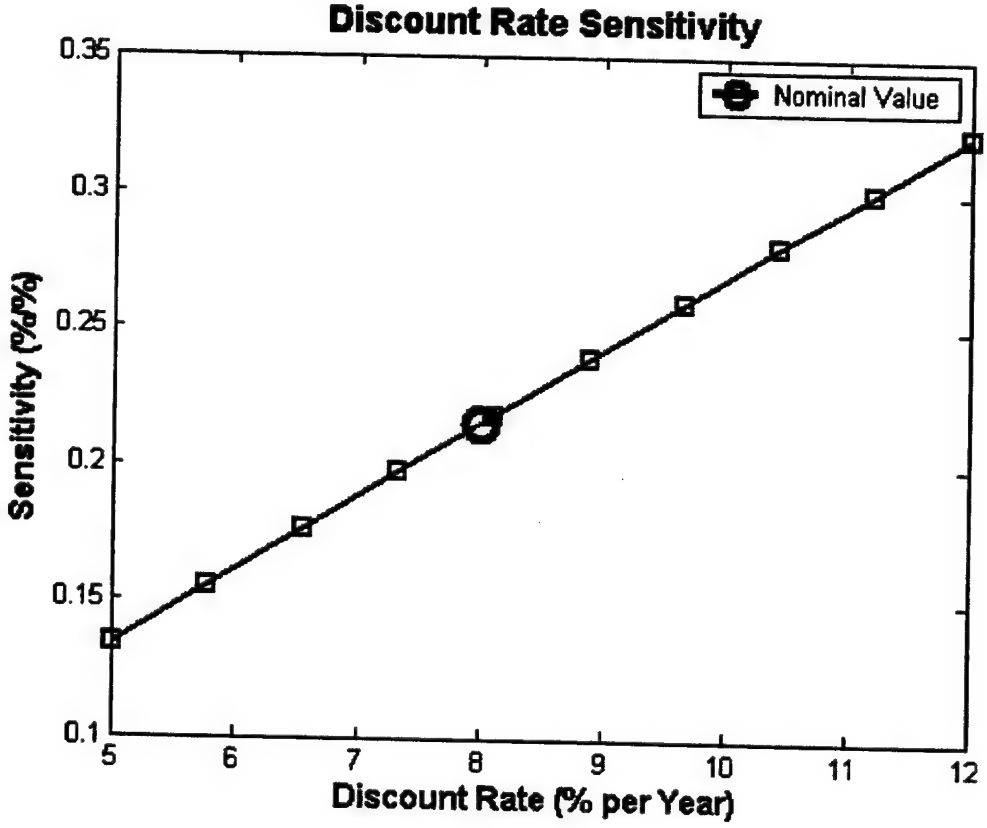


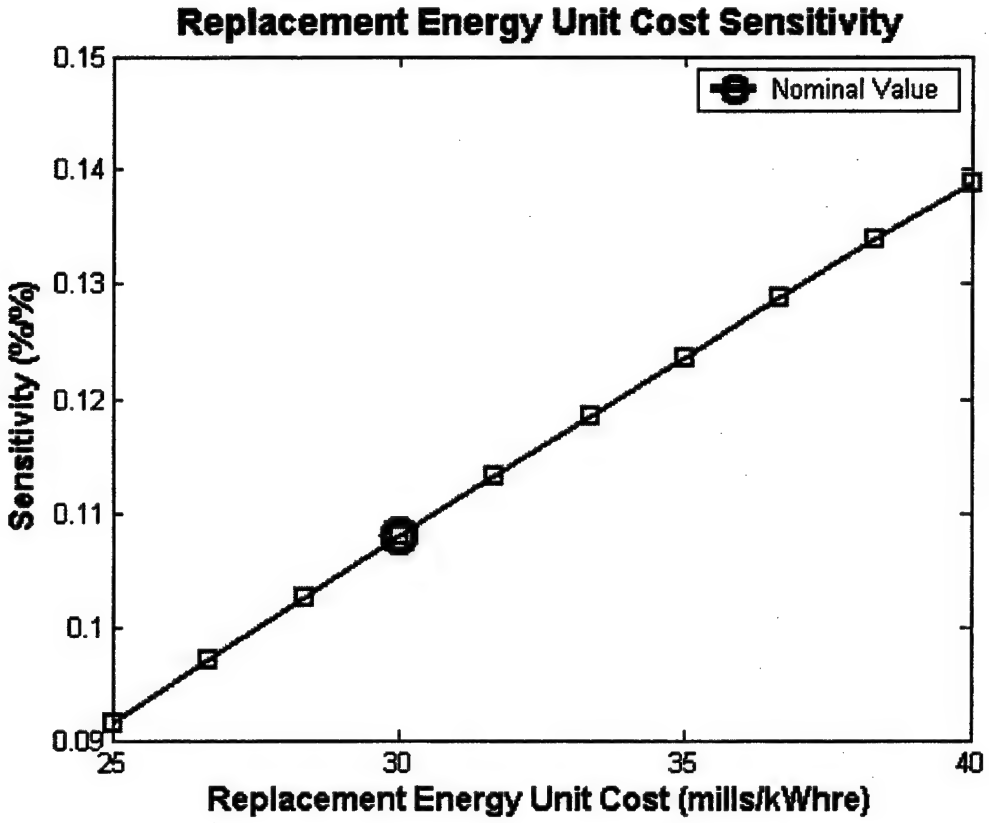
[e] Total cost includes levelized fuel cycle cost, shutdown O&M – including replacement energy costs, and the disposal fee.
[f] I.e. all parameters are at their reference condition.

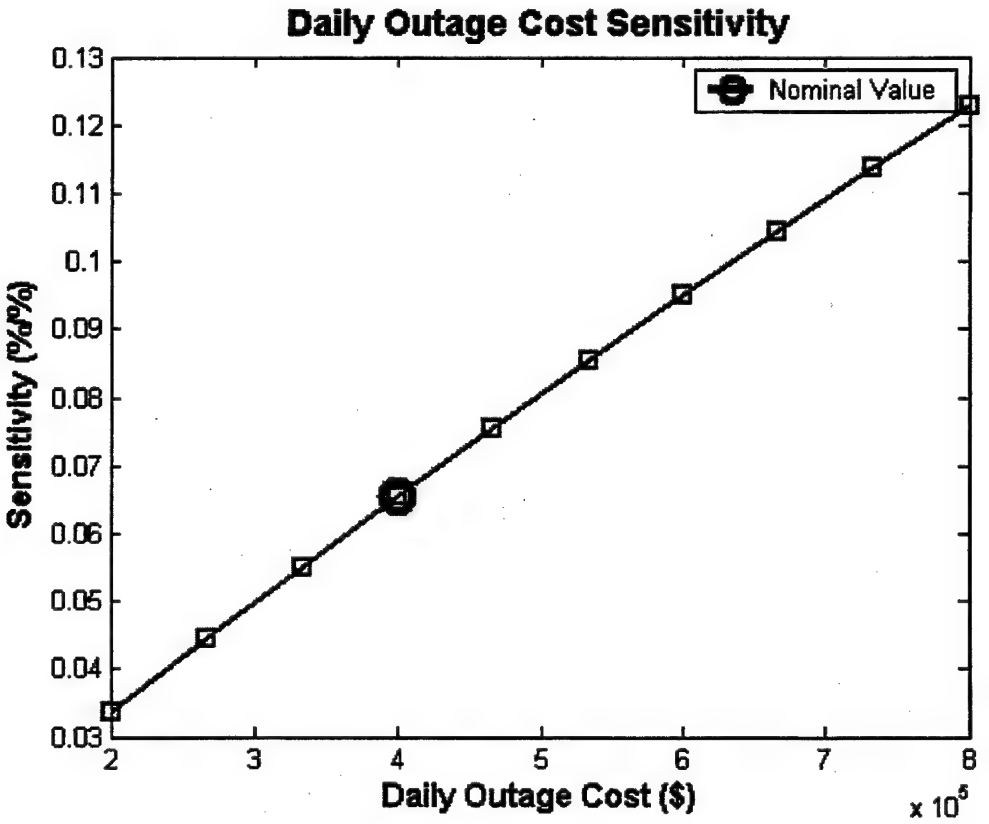

Uranium Conversion Price Sensitivity
Nominal Value
Sensitivity (%d%)
0.035
0.03
0.025
0.02
0.015
0.01
2
2.5
3
3.5
4
4.5
5
5.5
6
Uranium Conversion Price ($/kg Natural Uranium)

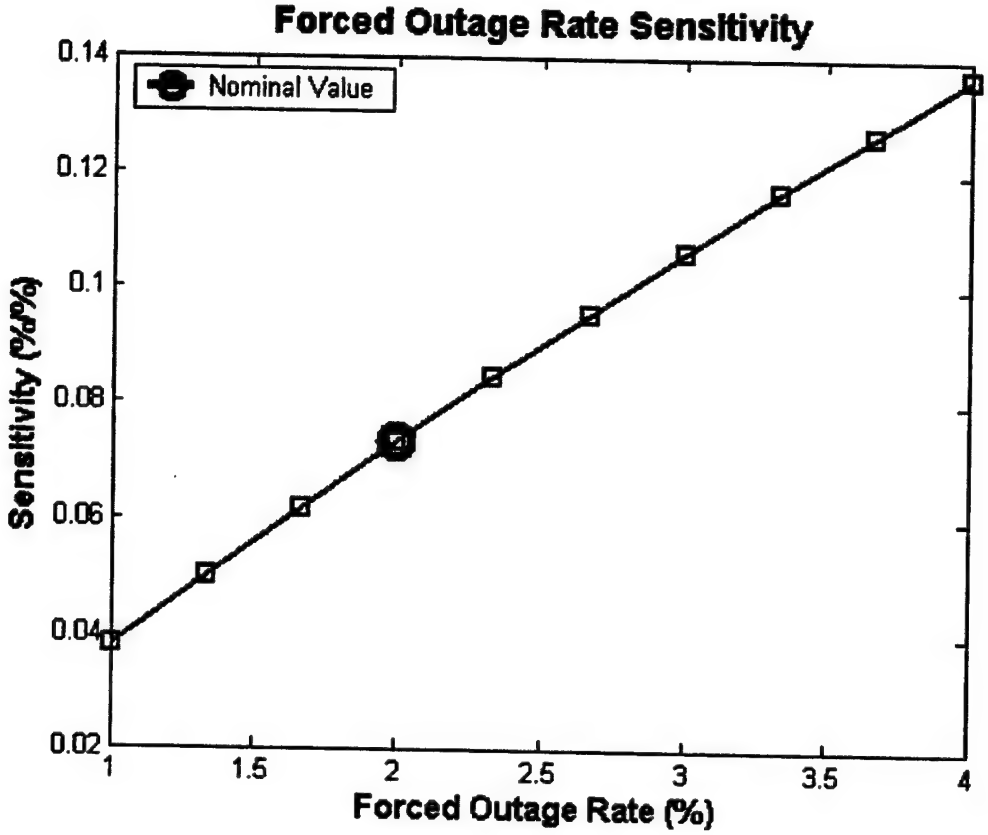

Enrichment Cost Sensitivity
Nominal Value
Sensitivity (%d%)
0.47
0.46
0.45
0.44
0.43
0.42
0.41
0.4
0.39
0.38
100
105
110
115
120
125
130
Enrichment Cost ($/kg SWU)

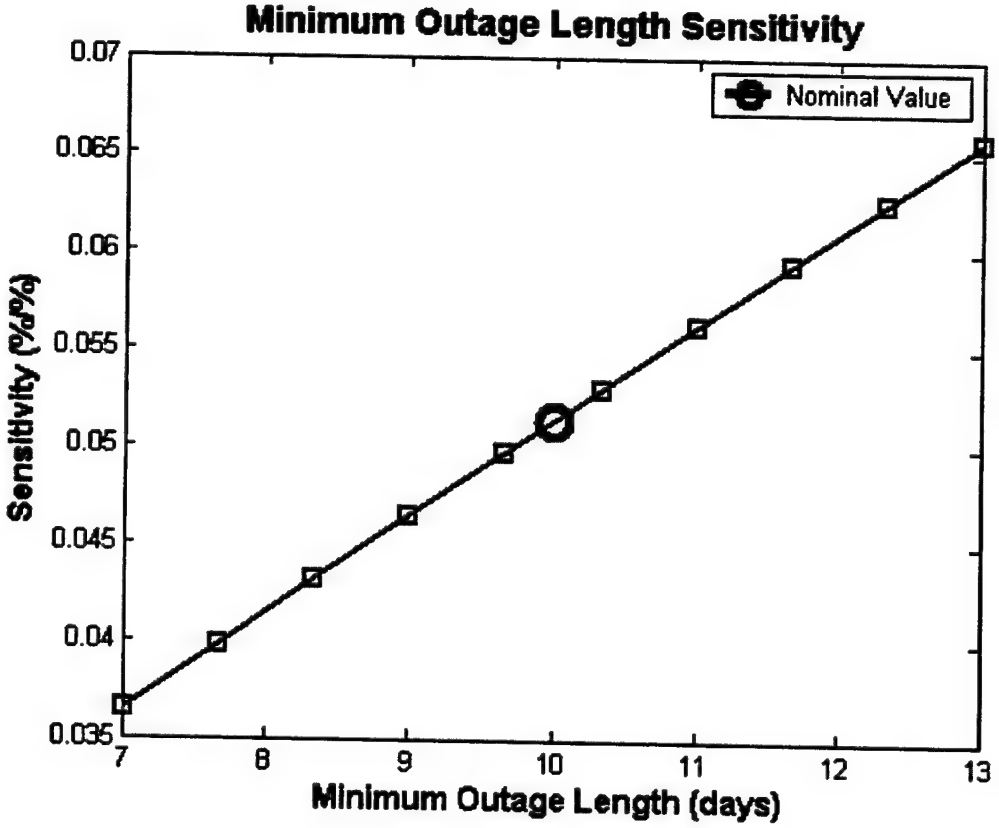

Fuel Fabrication Cost Sensitivity
Nominal Value
Sensitivity (%/%)
0.14
0.13
0.12
0.11
0.1
190
200
210
220
230
240
250
260
270
280
Fuel Fabrication Cost ($/kg Fuel)

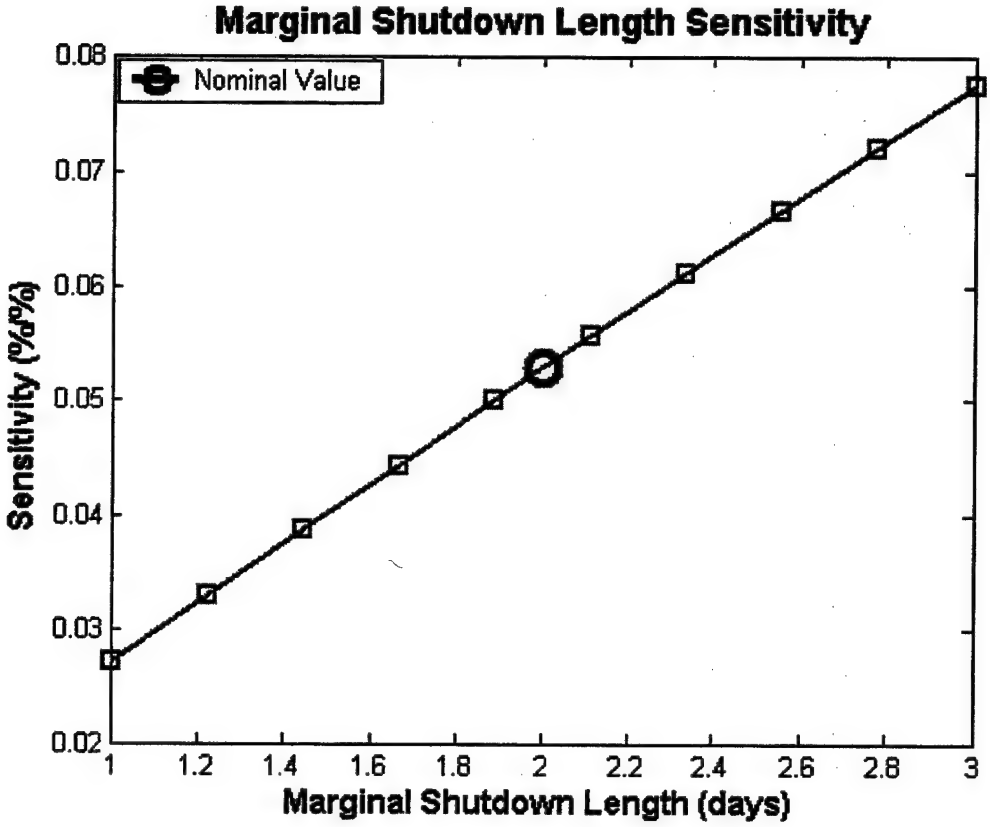

Discount Rate Sensitivity
Nominal Value
Sensitivity (%/%)
0.35
0.3
0.25
0.2
0.15
0.1
5
6
7
8
9
10
11
12
Discount Rate (% per Year)


Replacement Energy Unit Cost Sensitivity
Nominal Value
Sensitivity (%/%)
0.15
0.14
0.13
0.12
0.11
0.1
0.09
25
30
35
40
Replacement Energy Unit Cost (mills/kWhre)


Daily Outage Cost Sensitivity
Nominal Value
Sensitivity (%/%)
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
2
3
4
5
6
7
8
Daily Outage Cost ($)
x 10^5


Forced Outage Rate Sensitivity
Nominal Value
Sensitivity (%/%)
Forced Outage Rate (%)
0.14
0.12
0.1
0.08
0.06
0.04
0.02
1
1.5
2
2.5
3
3.5
4


Minimum Outage Length Sensitivity
Nominal Value
Sensitivity (%/%)
Minimum Outage Length (days)
0.07
0.065
0.06
0.055
0.05
0.045
0.04
0.035
7
8
9
10
11
12
13



For the values of the parameters used in this study, it is seen that the enrichment costs, uranium purchase costs, daily outage costs and discount rate have the largest effect on the overall fuel cycle cost.

One difficulty that is presented with this sensitivity study, is that the magnitude of the sensitivity for a given parameter is dependent upon the remaining parameters in the model. For this simple sensitivity study, all other model parameters are held at their best-guess values. The problem is that the magnitude of many of the parameter sensitivities is dependent upon the other variables. The fuel cycle cost is a function of several variables. The sensitivity analysis procedure discussed above is akin to taking the partial derivative of the fuel cycle cost function with respect to just one of the variables. This partial derivative has many different values, depending upon where within the n-dimensional domain of the fuel cycle cost function the derivative is evaluated. As an example of this weakness, the fuel cycle enrichment cost sensitivity is shown with assumed discount rates ranging from 6% to 11%.

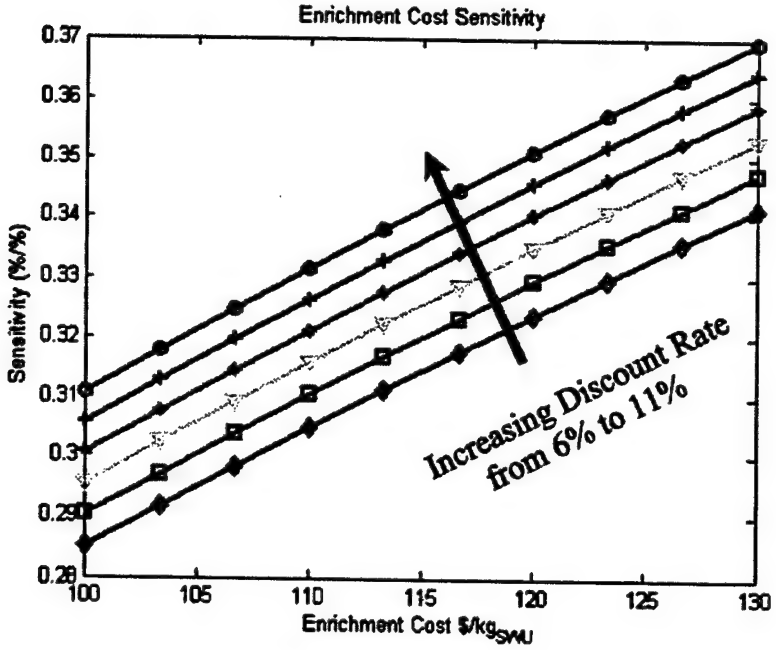


Figure 48:Illustration of Variability in Sensitivity.

There is no way that this difficulty can be eliminated. Furthermore, even if there was, any methodology that could be used to determine the overall model uncertainty (making use of this sensitivity data) would be unable to capture this behavior. This is the reason why the MCUP provides more accurate results than the ITDP.[g]

Uncertainty Analysis

It is a fact that all of the parameters under consideration in this study are subject to uncertainty. In order to deal with this uncertainty in a logical and consistent fashion, the following approach was used which is based on the approach used in reference [51]. A selection of parameters were assigned a non-parametric probability distribution function (PDF). This PDF is formed by selecting a most likely value (mode) for a given parameter and a minimum and maximum value that the parameter is 'reasonably' expected to take. In this context, the phrase 'reasonably' should be interpreted to mean: subject to the judgement of a panel of experts. For this analysis, no such panel was assembled, rather the judgment of the author was used in order to arrive at plausible values.

[g] The Monte Carlo Uncertainty Procedure (MCUP) and the Improved Thermal Design Procedure (ITDP) are two hot-channel analysis methodologies that were used in a thermal-hydraulic analysis of the IRIS core.

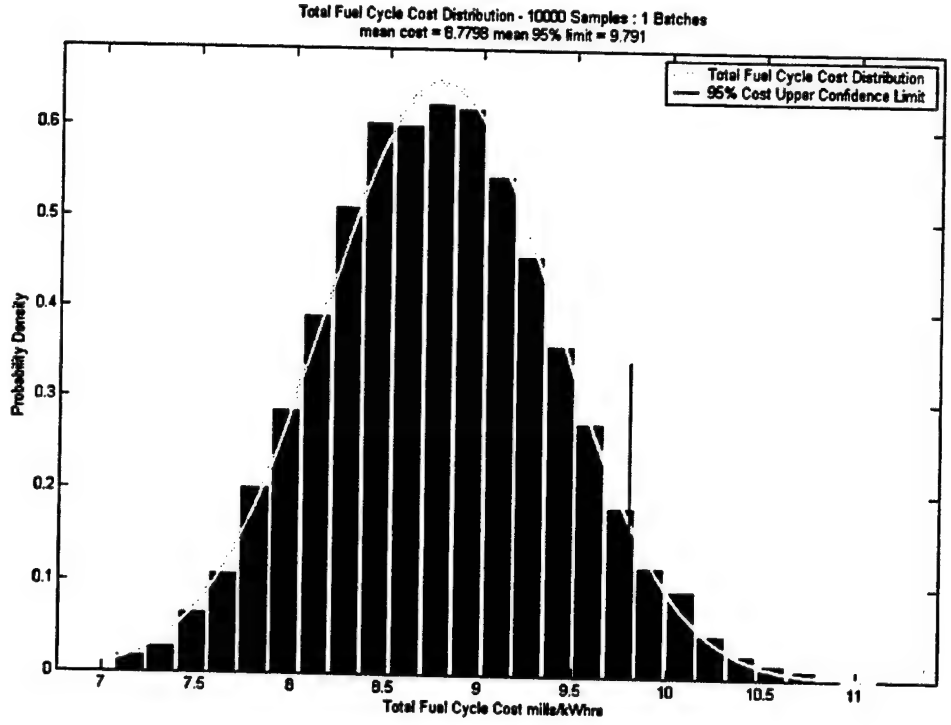
Once the distributions were created for each parameter under consideration, these distributions were sampled ten thousand times (each), the cost model was updated with the new sample parameters, and the total cost was computed. The statistical properties of the output total fuel cycle costs were then analyzed. The total cost mean value as well as the 5^{th} and 95^{th} percentile were computed and plotted. This process was repeated for each of the six fuel management strategies (number of batches).

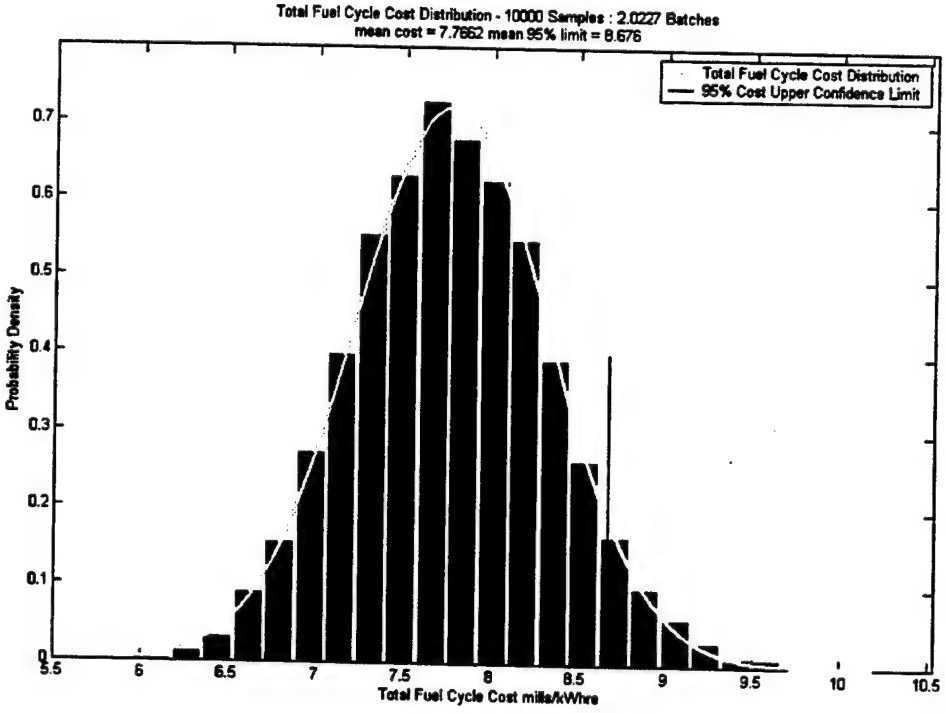
The characteristics of the parameter distributions are given in Table 28.

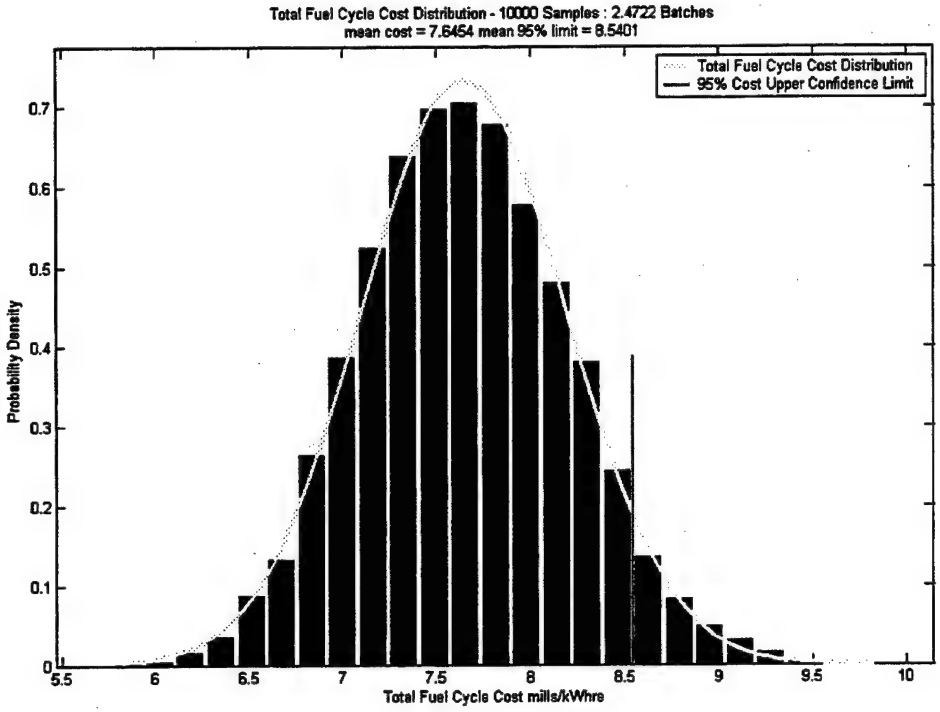
Parameter	Minimum	Mode	maximum
*Uranium Purchase Price ($/kgU)	21.7	27.7	50.6
*Uranium Conversion Price ($/kgU)	3.7	4.9	6.8
*Enrichment Price ($/SWU)	78.5	108	127.7
*Fuel Fabrication Price ($/kgU)	145.4	200	254.6
Discount Rate (%)	5	8	11
Minimum Shutdown Length (days)	7	10	13
Marginal Shutdown Length (days)	1	2	3
Daily Shutdown Cost ($/day)	$ 200 K	$ 400 K	$ 800 M
Forced Outage Rate (%)	1 %	2 %	3 %
Replacement Energy Cost (mills/kWhr)	24	30	36
* Based on percentage above or below Mode used in reference [51]. All others are based on author's best-guess.			

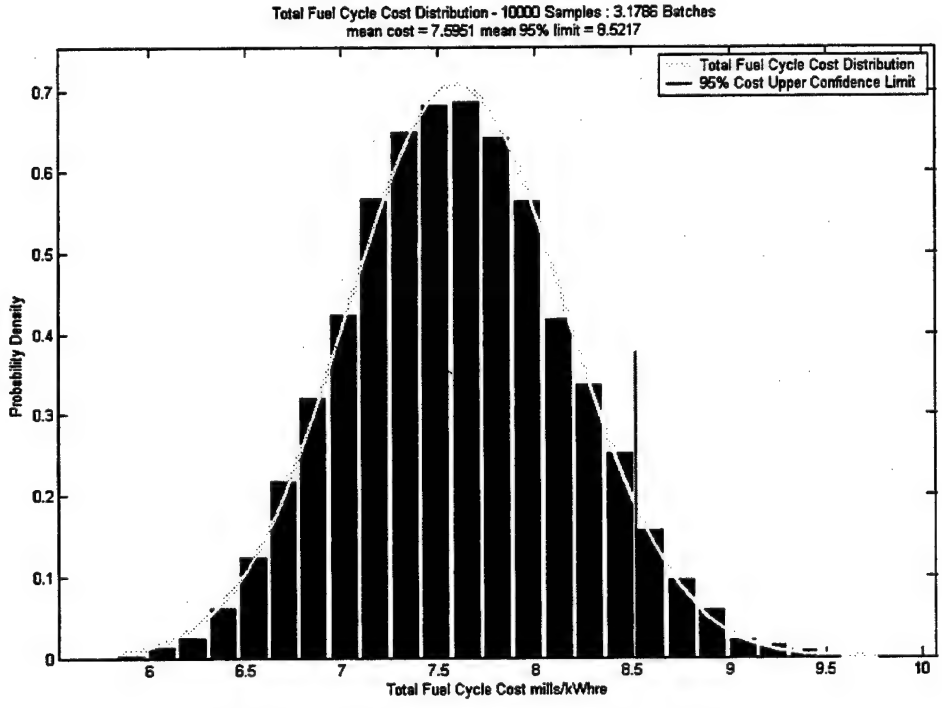
Table 28: Distributions Used for Monte Carlo Uncertainty Analysis of Fuel Cycle Costs.

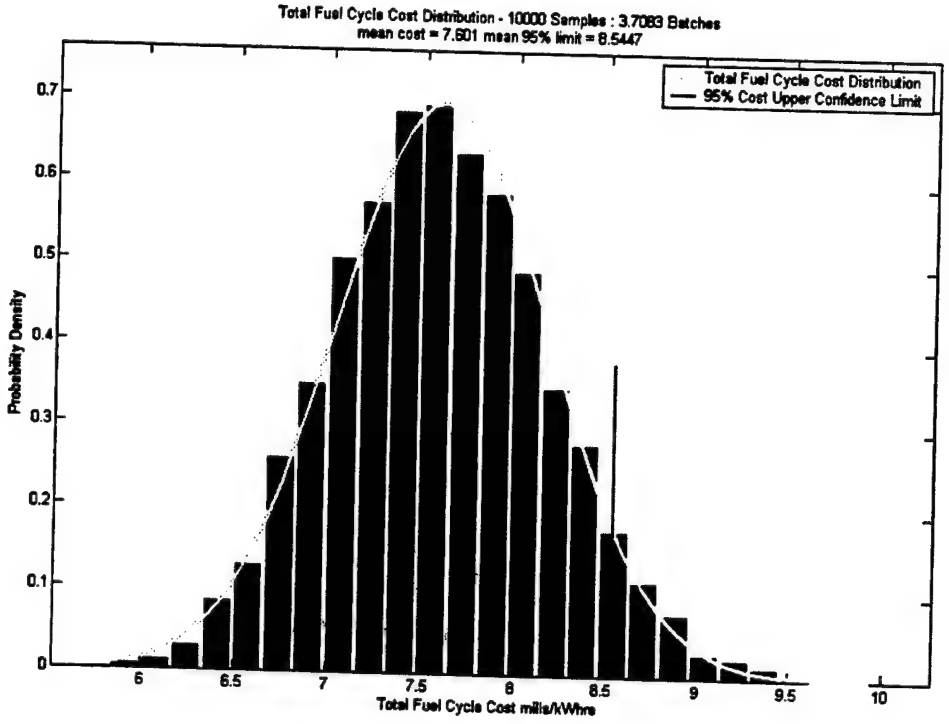
The results are presented here:

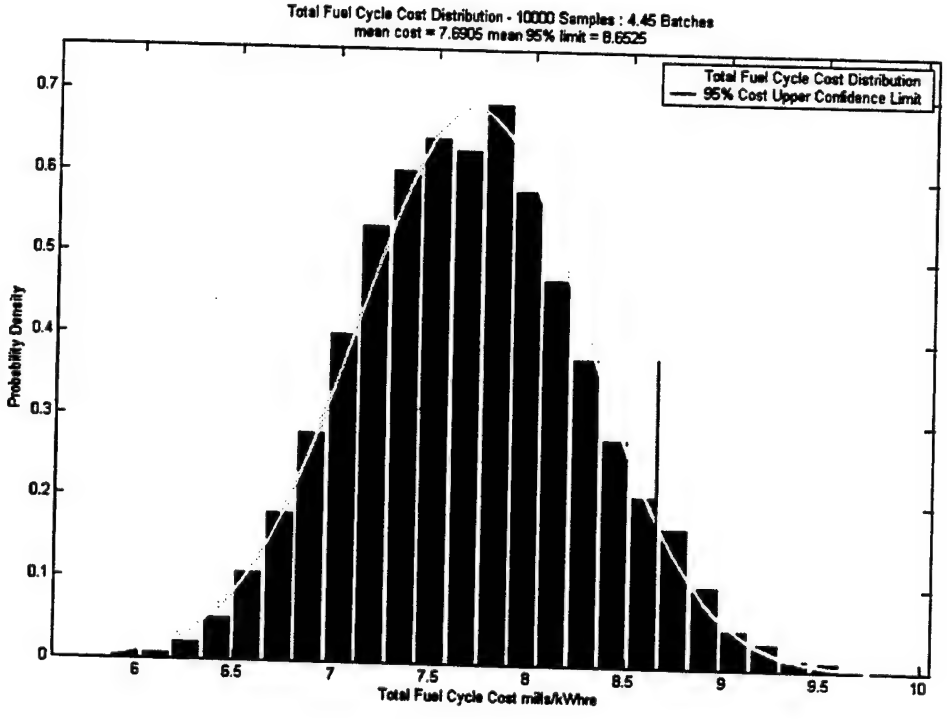

Total Fuel Cycle Cost Distribution - 10000 Samples : 1 Batches
mean cost = 8.7798 mean 95% limit = 9.791
Total Fuel Cycle Cost Distribution
95% Cost Upper Confidence Limit
Probability Density
0
0.1
0.2
0.3
0.4
0.5
0.6
7
7.5
8
8.5
9
9.5
10
10.5
11
Total Fuel Cycle Cost mills/kWhre


Total Fuel Cycle Cost Distribution - 10000 Samples : 2.0227 Batches
mean cost = 7.7662 mean 95% limit = 8.676
Total Fuel Cycle Cost Distribution
95% Cost Upper Confidence Limit
Probability Density
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10
10.5
Total Fuel Cycle Cost mills/kWhre


Total Fuel Cycle Cost Distribution - 10000 Samples : 2.4722 Batches
mean cost = 7.6454 mean 95% limit = 8.5401
Total Fuel Cycle Cost Distribution
95% Cost Upper Confidence Limit
Probability Density
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
5.5
6
6.5
7
7.5
8
8.5
9
9.5
10
Total Fuel Cycle Cost mills/kWhre


Total Fuel Cycle Cost Distribution - 10000 Samples : 3.1786 Batches
mean cost = 7.5951 mean 95% limit = 8.5217
Total Fuel Cycle Cost Distribution
95% Cost Upper Confidence Limit
Probability Density
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
6
6.5
7
7.5
8
8.5
9
9.5
10
Total Fuel Cycle Cost mills/kWhre


Total Fuel Cycle Cost Distribution - 10000 Samples : 3.7083 Batches
mean cost = 7.601 mean 95% limit = 8.5447
Total Fuel Cycle Cost Distribution
95% Cost Upper Confidence Limit
Probability Density
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
6
6.5
7
7.5
8
8.5
9
9.5
10
Total Fuel Cycle Cost mills/kWhre


Total Fuel Cycle Cost Distribution - 10000 Samples : 4.45 Batches
mean cost = 7.6905 mean 95% limit = 8.6525
Total Fuel Cycle Cost Distribution
95% Cost Upper Confidence Limit
Probability Density
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
6
6.5
7
7.5
8
8.5
9
9.5
10
Total Fuel Cycle Cost mills/kWhre

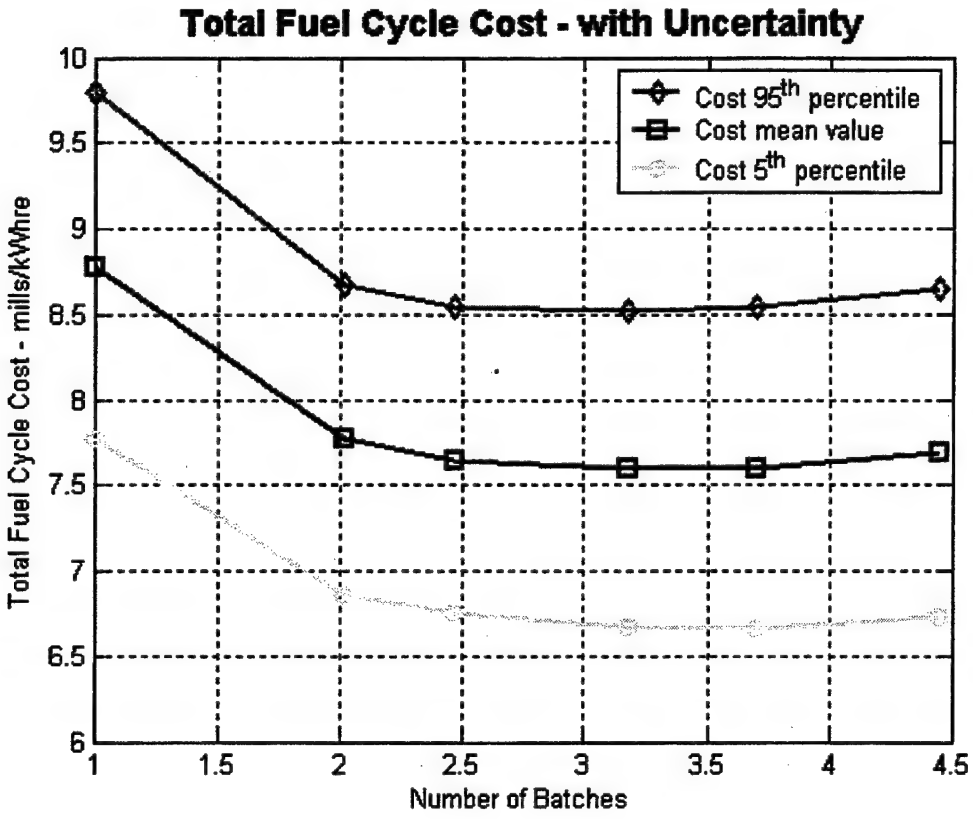


Figure 49: Total Fuel Cycle Cost Monte Carlo Analysis Results.

It is notable that, for many of the batches, the results are slightly skewed from normal. This is judged to be due to the non-symmetry of the probability distributions of some of the important parameters, notably: fuel enrichment unit cost, and daily refueling outage costs.

Conclusions

The results of this analysis indicate that a fuel management strategy where 2 to 3.5 batches are used would be the most economically attractive from a fuel cycle cost perspective. Higher costs are expected to occur for the single-batch 'straight burn' option, and also for strategies that involve more than 4 batches. These results are consistent with the common industry pratice of using a 2 to 3-batch fuel management strategy.

It is emphasized, however there is a considerable amount of uncertainty inherent in some of the model parameters used, particularly those parameters pertaining to operation

and maintenance (O&M) costs. While reasonably direct and unambiguous data[h] exist with regards to the front-end costs, nothing of the sort is available for the O&M costs. It is expected that the O&M costs for the IRIS core will be positively affected by reduced requirements for:

- Weld inspections (fewer welds are required due to the integral vessel design),
- Reactor coolant pump inspection (maintenance-free spool pumps are adopted)

In addition, ongoing efforts to optimize maintenance on critical plant components[52] (e.g. Turbine Generator Governor testing, pressure relief valve testing, main condenser waterbox cleaning) are expected to reduce the downtime requirement for shutdown inspection and maintenance. However, this potential cost reduction has not been accounted for in the present analysis. If the assumed values are greatly in error, the impact on the results could be significant: on the order of 2 – 3 mills/kWhre and, because of the flat shape of the curve in Figure 49 could completely change the resulting optimum fuel management strategy. At the same time, the absolute difference in the cost will likely remain small, and therefore the impact of using a non-optimum scheme may be acceptable.

Matlab Code

To document the calculations done, the Matlab code used for the analysis is provided here.

All of the parameters used in the econmic model are stored in a Matlab 'struct array'.[i] This data structure is initialized with the script below:

```
%initialize_model.m

% ===== ASSUMED/ASSIGNED PARAMETERS ====================================

model.thermal_power = 1000; %MWth
model.electric_power = 335; %MWe
model.assemblies = 89;

% FUEL_RODS_PER_ASSEMBLY = 264;
% ACTIVE_CORE_HEIGHT = 14;
```

[h] As well as specification by the OECD guidelines in the case of front-end fuel cycle costs.
[i] This data type is similar in syntactically and semantically similar to the data structure (called 'struct') in the C programming language.

```
% AVERAGE_LINEAR_POWER = 3.04; %kW/ft
% PELLET_DIAMETER = 0.3225; %in
% FUEL_DENSITY = 0.96; %fraction of theoretical density
% FUEL_DISHING = .01; %fraction dishing
% HM_PER_ASSEMBLY = 545.1; %kgU/FA
model.fuel_loading= 48.5e3; %kg U/core
model.forced_outage_rate = 0.02;
%model.refueling_outage_length = 30; %days
model.feed_enrichment = 0.00711; %weight fraction of U-235
model.tails = 0.003; %weight fraction of U-235 in tails
model.uranium_purchase_lead_time = 24; %months
model.uranium_conversion_lead_time = 18; %months
model.uranium_enrichment_lead_time = 18; %months
model.uranium_fabrication_lead_time = 12; %months
model.uranium_purchase_cost = 27.66; %$/kgU
model.uranium_conversion_cost = 4.90; %$/kgU
model.swu_cost = 108; %$/kgSWU
model.fuel_fabrication_cost = 200; %$/kgU
model.discount_rate = 0.08;
model.fuel_enrichment = 0.0495; %weight percent U-235 in fuel
model.replacement_energy_unit_cost = 30.0; %mills/kW-hr
model.outage_daily_cost = 1e6; %dollars
model.disposal_fee = 1.0; %mills/kWhre
model.single_batch_discharge_burnup = 38000; %MWd/tU
model.batches = 1;
model.min_shutdown_length =15; %days
model.variable_shutdown_length = 5; %days/burnup-year

% ===== ASSUMED/ASSIGNED PARAMETERS ======================================
```

The total fuel cycle cost is computed with the following function:

```
%calc_costs.m
function [lfcc,repl_energy_cost,disposal_fee,shutdown_o_and_m,capacity_factor] =
calc_costs(model);
% ===== ASSUMED/ASSIGNED PARAMETERS ======================================
THERMAL_POWER = model.thermal_power;
ELECTRIC_NET = model.electric_power;
ASSEMBLIES = model.assemblies;
FUEL_LOADING = model.fuel_loading;
FORCED_OUTAGE_RATE = model.forced_outage_rate;
FEED_ENRICHMENT = model.feed_enrichment;
TAILS = model.tails;
U_PURCHASE_LEAD_TIME = model.uranium_purchase_lead_time;
U_CONVERSION_LEAD_TIME = model.uranium_conversion_lead_time;
U_ENRICHMENT_LEAD_TIME = model.uranium_enrichment_lead_time;
U_FABRICATION_LEAD_TIME = model.uranium_fabrication_lead_time;
U_PURCHASE_COST = model.uranium_purchase_cost;
U_CONVERSION_COST = model.uranium_conversion_cost;
SWU_COST = model.swu_cost;
FUEL_ASSEMBLY_FABRICATION_COST = model.fuel_fabrication_cost;
DISCOUNT_RATE = model.discount_rate;
FRESH_FUEL_ENRICHMENT = model.fuel_enrichment;
REPLACEMENT_ENERGY_UNIT_COST = model.replacement_energy_unit_cost;
DAILY_LABOR_AND_MATERIALS_COST_OF_OUTAGE = model.outage_daily_cost;
SINGLE_BATCH_DISCHARGE_BURNUP = model.single_batch_discharge_burnup;
DISPOSAL_FEE = model.disposal_fee;
BATCHES = model.batches;
MIN_REFUELING_OUTAGE_LENGTH = model.min_shutdown_length;
VAR_SHUTDOWN_LENGTH = model.variable_shutdown_length;

% ===== ASSUMED/ASSIGNED PARAMETERS ======================================

specific_power = (1000*THERMAL_POWER)/FUEL_LOADING; %kWth/kgU
thermal_efficiency = ELECTRIC_NET/THERMAL_POWER;
plant_availability = 1.0-FORCED_OUTAGE_RATE;
```

```
num_fresh_fa_per_batch = ASSEMBLIES/BATCHES;
discharge_burnup = SINGLE_BATCH_DISCHARGE_BURNUP*(2*BATCHES)/(1+BATCHES);%MWd/tU
burnup_per_batch = discharge_burnup/BATCHES; %MWd/tU-batch
daily_burnup = specific_power*plant_availability; %MWd/tU-day
burnup_length_per_batch = burnup_per_batch/daily_burnup; %days
REFUELING_OUTAGE_LENGTH = MIN_REFUELING_OUTAGE_LENGTH +
(VAR_SHUTDOWN_LENGTH*burnup_length_per_batch/365.25);
refueling_cycle_length = REFUELING_OUTAGE_LENGTH + burnup_length_per_batch; %days
capacity_factor = plant_availability*(1 -
REFUELING_OUTAGE_LENGTH/refueling_cycle_length);
kwhre_per_batch = 1000*ELECTRIC_NET*capacity_factor*burnup_length_per_batch*24; %kWhre
cost_per_shut_down = DAILY_LABOR_AND_MATERIALS_COST_OF_OUTAGE* ...
    REFUELING_OUTAGE_LENGTH; %dollars
fuel_residence_time = refueling_cycle_length*BATCHES;
F_P = (FRESH_FUEL_ENRICHMENT - TAILS)/(FEED_ENRICHMENT - TAILS);
V_p = (2*FRESH_FUEL_ENRICHMENT - 1)*log(FRESH_FUEL_ENRICHMENT/(1-FRESH_FUEL_ENRICHMENT));
V_f = (2*FEED_ENRICHMENT - 1)*log(FEED_ENRICHMENT/(1-FEED_ENRICHMENT));
V_t = (2*TAILS - 1)*log(TAILS/(1-TAILS));
F = F_P * FUEL_LOADING; %kg - mass of feed material needed
SWU = (V_p-V_t) - ((FRESH_FUEL_ENRICHMENT - TAILS)/(FEED_ENRICHMENT - TAILS))*(V_f -
V_t);
start_of_irr_ore_cost =
(F_P)*(U_PURCHASE_COST)*(1+DISCOUNT_RATE*(U_PURCHASE_LEAD_TIME/12));
start_of_irr_conv_cost =
(F_P)*(U_CONVERSION_COST)*(1+DISCOUNT_RATE*(U_CONVERSION_LEAD_TIME/12));
start_of_irr_enrich_cost = SWU*(SWU_COST)*(1+DISCOUNT_RATE*(U_ENRICHMENT_LEAD_TIME/12));
start_of_irr_fab_cost = (FUEL_ASSEMBLY_FABRICATION_COST) *
(1+DISCOUNT_RATE*(U_FABRICATION_LEAD_TIME/12));
I_o = start_of_irr_ore_cost + start_of_irr_conv_cost + start_of_irr_enrich_cost + ...
    start_of_irr_fab_cost;
lfcc = (I_o * DISCOUNT_RATE
)/(8.766*specific_power*capacity_factor*thermal_efficiency*...
    (1-exp(-DISCOUNT_RATE*(fuel_residence_time/365.25))));

required_replacement_energy = (ELECTRIC_NET * 1000) * refueling_cycle_length * (1 -
capacity_factor) * (24);
annualized_replacement_energy_cost = required_replacement_energy *
((REPLACEMENT_ENERGY_UNIT_COST/1000)/...
    (refueling_cycle_length/365.25));%last part converts days to years
ann_energy_cost = annualized_replacement_energy_cost;
repl_energy_cost = (1-capacity_factor)*REPLACEMENT_ENERGY_UNIT_COST; %mills/kwhre
shutdown_o_and_m = (cost_per_shut_down/kwhre_per_batch)*(1000); %mills/kwhre
disposal_fee = DISPOSAL_FEE;
```

The Monte Carlo analysis requires the ability to 'invert' the triangular probability distributions used for the uncertain parameters, in the sense that, in effect, a cumulative distribution function is created. The distributions were described only by the mode, minimum value and maximum value. Based on this information, and the knowledge that the distribution is triangular, a function is required that, when given a number in the range of 0 to 1, a value is returned that corresponds to the result from passing the input number to the cumulative distribution function. This is required to provide the parameter sample values for the Monte Carlo Analysis. The function to perform this task is provided here:

```
%invert_tri_dist.m
```

```
function val = invert_tri_dist(tri_dist,prob)

h = 2/(tri_dist.high - tri_dist.low);

area_to_mode = (1/2)*(tri_dist.mode - tri_dist.low)*h;
slope_up = h/(tri_dist.mode - tri_dist.low);
slope_down = h/(tri_dist.high - tri_dist.mode);

if (prob == area_to_mode)
    val = tri_dist.mode
elseif(prob < area_to_mode)
        val = sqrt(2*prob/slope_up) + tri_dist.low;
else %prob > area_to_mode
     val = -sqrt(2*(1-prob)/slope_down) + tri_dist.high;
end
```

With the above function, the resulting Monte Carlo Analysis script is fairly straight-forward:

```
%mc_fcc_analysis.m
initialize_model;
%==========================================================================
%==========================================================================
%================== Initialize the Probability Distributions ==============
%==========================================================================
%==========================================================================
% ==== same percentage change as KAERI paper ===========

u_cost_dist.mode = model.uranium_purchase_cost;
u_cost_dist.low = (1-.215)*u_cost_dist.mode;
u_cost_dist.high = (1.826)*u_cost_dist.mode;

u_conv_dist.mode = model.uranium_conversion_cost;
u_conv_dist.low = (1-.25)*u_conv_dist.mode;
u_conv_dist.high = (1.38)*u_conv_dist.mode;

u_enrich_dist.mode = model.swu_cost;
u_enrich_dist.low = (1-.2733)*u_enrich_dist.mode;
u_enrich_dist.high = (1.182)*u_enrich_dist.mode;

u_fab_dist.mode = model.fuel_fabrication_cost;
u_fab_dist.low = (1-.273)*u_fab_dist.mode;
u_fab_dist.high = (1.273)*u_fab_dist.mode;

% ======================================================

% ========== my swags ==================================

disc_rate_dist.mode = model.discount_rate;
disc_rate_dist.low = disc_rate_dist.mode - .03;
disc_rate_dist.high = disc_rate_dist.mode + .03;

sd_length_dist.mode = model.min_shutdown_length;
sd_length_dist.high = 20;
sd_length_dist.low = 10;

var_sd_length_dist.mode = model.variable_shutdown_length;
var_sd_length_dist.low = (1-0.5)*var_sd_length_dist.mode;
var_sd_length_dist.high = (1.5)*var_sd_length_dist.mode;

var_sd_cost_dist.mode = model.outage_daily_cost;
var_sd_cost_dist.high = (2)*var_sd_cost_dist.mode;
var_sd_cost_dist.low = (0.5)*var_sd_cost_dist.mode;

repl_eng_cost_dist.mode = model.replacement_energy_unit_cost;
repl_eng_cost_dist.low = (1 - 0.20)*repl_eng_cost_dist.mode;
```

```
repl_eng_cost_dist.high = (1.2) * repl_eng_cost_dist.mode;

forced_out_rate_dist.mode = model.forced_outage_rate;
forced_out_rate_dist.low = (1 - 0.5)*forced_out_rate_dist.mode;
forced_out_rate_dist.high = (1.5)*forced_out_rate_dist.mode;

% ================================================
%=========================================================================
%=========================================================================
%=========================================================================
%=========================================================================
all_batches = [1 (89/44) (89/36) (89/28) (89/24) (89/20)];
for batch_space = 1:6
    model.batches = all_batches(batch_space);

    NUM_SAMPLES = 10000;
    sample_lfcc = zeros(NUM_SAMPLES,1);
%   rand('state',0); %for de-bugging
    samples = rand(NUM_SAMPLES,11);

    for j = 1:NUM_SAMPLES
        model.uranium_purchase_cost = invert_tri_dist(u_cost_dist,samples(j,1));
        model.uranium_conversion_cost = invert_tri_dist(u_conv_dist,samples(j,2));
        model.swu_cost = invert_tri_dist(u_enrich_dist,samples(j,3));
        model.fuel_fabrication_cost = invert_tri_dist(u_fab_dist,samples(j,4));
        model.discount_rate = invert_tri_dist(disc_rate_dist,samples(j,5));
        model.replacement_energy_unit_cost =
invert_tri_dist(repl_eng_cost_dist,samples(j,7));
        model.forced_outage_rate = invert_tri_dist(forced_out_rate_dist,samples(j,8));
        model.min_shutdown_length = invert_tri_dist(sd_length_dist,samples(j,9));
        model.variable_shutdown_length =
invert_tri_dist(var_sd_length_dist,samples(j,10));
        model.outage_daily_cost = invert_tri_dist(var_sd_cost_dist,samples(j,11));
        sample_lfcc(j) = calc_costs_sens(model);
    end

    figure
    %now, analyze the results
    %normalized histogram
    [n,bin_cent] = hist(sample_lfcc,25);
    n = n .* (1/(length(sample_lfcc)*(bin_cent(2) - bin_cent(1))));
    bar(bin_cent,n);
    %format the plot
    max_occurance = max(n);
    min_center = min(bin_cent);
    max_center = max(bin_cent);
    x_span = max_center - min_center;
    x_margin = 0.1*x_span;
    x_min = min_center - x_margin;
    x_max = max_center + x_margin;
    y_min = 0;
    y_max = 1.1*max_occurance;
    axis([x_min x_max y_min y_max]);
    y_space = linspace(y_min,0.5*y_max,10);

    min_calc_lfcc = min(sample_lfcc);
    max_calc_lfcc =max(sample_lfcc);
    NUM_STEPS = 100;
    mean_lfcc(batch_space) = mean(sample_lfcc);
    std_lfcc(batch_space) = std(sample_lfcc);
    mean_95_lfcc(batch_space) = mean_lfcc(batch_space) + 1.645*(std_lfcc(batch_space));
    mean_05_lfcc(batch_space) = mean_lfcc(batch_space) - 1.645*(std_lfcc(batch_space));
    X = linspace(min_calc_lfcc,max_calc_lfcc,NUM_STEPS);
    y = normal_dist(mean_lfcc(batch_space),std_lfcc(batch_space),X);

    hold on
    plot(X,y,'-c','LineWidth',3)

    tit_str = {['Total Fuel Cycle Cost Distribution - ' num2str(NUM_SAMPLES) ' Samples : '
num2str(model.batches) ' Batches'], ...
```

```
          ['mean cost = ' num2str(mean_lfcc(batch_space)) ' mean 95% limit = '
num2str(mean_95_lfcc(batch_space))];
    title(tit_str);
    xlabel('Total Fuel Cycle Cost mills/kWhre');
    ylabel('Probability Density');
    mean_plot = ones(10,1) .* mean_95_lfcc(batch_space);
    plot(mean_plot,y_space,'-m','LineWidth',2);
    legend('Total Fuel Cycle Cost Distribution','95% Cost Upper Confidence Limit')

end

figure
plot(all_batches,mean_95_lfcc,'-dr','LineWidth',2);
hold on
plot(all_batches,mean_lfcc,'-sb','LineWidth',2);
plot(all_batches,mean_05_lfcc,'-og','LineWidth',2);
title('Total Fuel Cycle Cost - with Uncertainty','FontSize',14,'FontWeight','bold');
legend('Cost 95^{th} percentile','Cost mean value','Cost 5^{th} percentile')
ylabel('Total Fuel Cycle Cost - mills/kWhre');
xlabel('Number of Batches');
```

Note that this function makes use of a utility function normal_dist.m. This function is used to create plot points for a normal curve with a specified mean and standard deviation. This is useful for providing the superimposed normal curves that are used in visualizing the Monte Carlo Analysis results.

```
%normal_dist.m

function y = normal_dist(mean,std,x);
%function y = normal_dist(mean,std,x) takes as arguements the mean and standard deviation
%of a proposed normal distribution.  Also, a vector x - representing the discretized
%domain (that part of the pdf to be plotted)

y = (1/(sqrt(2*pi)*std)) .* exp(-(1/2).*((x - mean)/ std).^2);
```